Tampere University

Eetu Friman

# BUILDING AND ADAPTING SYSML BASED SYSTEM ARCHITECTURE FRAMEWORK FOR MECHATRTONIC SYSTEMS

Faculty of Engineering and Natural Sciences
Master's thesis
26.10.2020

# ABSTRACT

Eetu Friman: Building and adapting a SysML based system architecture framework for mechatronic systems.
Master's thesis
Tampere University
Mechanical Engineering M.Sc.
10 / 2020

---

Implementing a SysML based systems engineering strategy to a project can be challenging as readily available methodologies and modeling guidelines are sparse. Lack of guidelines and modeling frameworks is one of the main reasons SysML faces resistance in the industry. Existing methods have to be tailored to suit project specific needs and to accommodate SysML, without a good starting point this can involve substantial amount of work and poses risks to companies.

This thesis sets out to study SysML application on mechatronics and derive an architecture framework that acts as a guideline and a structure for SysML modeling. The framework presented in this work is created based on literature findings and prototyped on a case study. The framework is built to be general, process independent and easy to implement, offering several tools to be applied to a mechatronics design process.

Prior studies have identified some of the key issues SysML faces in a mechatronic setting. Several solutions to these problems have been given but no exhaustive answer has been provided. This work combines aspects of prior studies into one framework and tries to create a good starting point to adapting SysML into a design process.

Keywords: SysML, Architecture framework, Mechatronics, Systems engineering

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

This project has received funding from the European Union's Horizon2020 research and innovation programme under grant agreement No. 820971

# TIIVISTELMÄ

Eetu Friman: SysML pohjaisen järjestelmäarkkitehtuurikehyksen rakentamien ja sovittamien mekatronisille järjestelmille.
Diplomityö
Tampereen yliopisto
Konetekniikka DI
10 / 2020

---

SysML pohjaisen mallintamisen käyttöönotto järjestelmäsuunnittelussa voi olla haastavaa saatavilla olevien metodien ja suositusten puutteessa. Näiden suositusten ja mallinnuskehysten puute on yksi pääsyistä sille miksi SysML ei ole teollisuudessa laajemmassa käytössä. Olemassa olevat metodit täytyy räätälöidä kullekin prosessille sopivaksi, ilman hyvää lähtökohtaa tästä saattaa koitua suuri määrää työtä ja riskejä yritystoiminnalle.

Tämän työn tavoitteena on rakentaa ja esitellä arkkitehtuurin mallintamiseen soveltuva mallinnuskehys, joka toimii ohjeistuksena ja rakenteena SysML mallinnukselle. Tässä työssä esitelty Mallinnuskehys on rakennettu kirjallisuuden suositusten ja case-studyn perusteella. Mallinnuskehys on suunniteltu yleispäteväksi ja helposti sovellettavaksi, tarjoten useita työkaluja mekatronisen järjestelmän arkkitehtuurin mallintamiseen.

Useat tutkimukset ovat tunnistaneet erinäisiä haasteita, jotka tulee huomioida, kun SysML:ää sovelletaan mekatronisten järjestelmien mallintamiseen. Yksittäisiin haasteisiin on tarjottu ratkaisuja, mutta yhtenäistä lopullista vastausta ei ole saatavilla. Tämä työ pyrkii yhdistämään yksittäiset ratkaisut yhtenäiseksi kokonaisuudeksi tarjoten hyvän lähtökohdan SysML:n käyttöönottoon.

Avainsanat: SysML, Arkkitehtuuri, Mekatroniikka, Järjestelmäsuunnittelu

# PREFACE

It has been quite an interesting year.

Tampere, 26 October 2020

Eetu Friman

# CONTENTS

# LIST OF FIGURES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| SysML | Systems Modelling Language |
| UML | Unified Modelling Language |
| SE | Systems Engineering |
| RE | Requirements engineering |
| MBSE | Model Based Systems Engineering |
| MBRE | Model Based Requirements Engineering |
| OMG | Object Management Group |
| INCOSE | International Council on Systems Engineering |
| EA | Enterprise Architect |
| SAM | System Architecture Model |
| ACRE | Approach to Context-based Requirements Engineering |
| SOI | System of Interest |
| V&V | Verification and Validation |

# 1. INTRODUCTION

Development of a mechatronic system is an interdisciplinary task that requires tight collaboration between different design domains. When system complexity increases and multiple disciplines are present, communication and well-defined system architecture become crucial. If each discipline uses their domain specific jargon and tools to describe the system, communication issues arise, and it will become difficult to form a coherent picture of the system as a whole. System architecture should be defined in a discipline neutral language so that it can be used as a design reference and to enable interdisciplinary communication and collaboration.

SysML (Systems Modeling Language) is slowly gaining popularity in the industry as a domain neutral architecture description language. SysML can be used to create one architecture model that integrates all design domains, hence closing the gap between them. When the whole system can be viewed as a single entity, issues such as system integration, design coverage and change management can be addressed. In order to create a coherent model that covers the whole system, guidelines should be used to make sure that all aspects of the system are considered in the model. Without modeling guidelines, the model easily becomes hard to navigate and read, this makes finding potential errors harder [1]. This issue is amplified when multiple people are working on the same model. There are several ways a system can be presented in the model, in order to guarantee a coherent model, conventions must be in place so that everyone working on the model follows the same modeling practices.

One of the main reasons why SysML adaptation faces resistance in the industry is that there are no readily available methodologies and guidelines that can be easily adapted to a design process [2] [3]. SysML does not require usage of specific design methodologies, but as a relatively new and complex language, resources have to be allocated to training the engineers to its usage. When SysML replaces some of the traditionally used tools, guidelines are needed to show when how and why certain SysML features should be used, this is where architecture frameworks become useful. An architecture framework specifies the modeling artifacts needed to define a system architecture, it can be seen as a template and a guideline for creating an architecture description. Architecture frameworks are commonly used for defining enterprise structures or military systems but can also be utilized to define architecture of an autonomous robot or any other physical system. The notion of a SysML based architecture framework is not new [4], but there are not may available and most are too general to be applied as is. As there are no suitable SysML based frameworks, the purpose of this thesis is to adapt and create a SysML based architecture framework that can be utilized in a design process of mechatronic systems.

# 2. RESEARCH OBJECTIVES, METHODOLOGY AND MATERIALS

This thesis is done as a case- and literature study. First SysML is studied based on existing research and literature. Found methodology, techniques and frameworks are then adapted and applied on a case-study. Based on the case study, suitable SysML usage modes are combined into a system architecture framework.

## 2.1 Research objectives and methods

The objective of this thesis is to explore SysML and its possibilities and to adapt a suitable architecture framework for usage on a design process of a mechatronic system. First a literature study is conducted on subjects regarding SysML and its usage in mechatronics, Systems engineering, and model-based systems engineering in order to find suitable guidelines, methods, tools and recommendations that can be used on modeling of mechatronic systems. Literature findings are utilized on a case-study where a robotic system is modeled based on the found methods and recommendations. Purpose of the case-study is to gather modeling data and information of the modeling process. The resulting model is then analyzed during and after the modeling process in order to identify and resolve key issues regarding modeling workflow, structure and organization. The found solutions and the key aspects of system architecture modeling are then combined into a framework that can be used as a modeling guideline on a design of an autonomous robot and mechatronic systems in general.

In order to create a SysML based system architecture framework these individual questions must be answered:

1. What are the most useful SysML usage modes for mechatronics design?
2. How SysML can be applied to different phases of a design process?
3. What are the key aspects that should be included in an architecture model?
4. How an architecture model should be structured?

## 2.2 Limitations and Assumptions

The case-study is done on Robominers research project. Usage of SysML in the project does not fully start during the first months, that is the time allocated for the study. This means that only preliminary phases of the project can be utilized in the case-study. The material available for study is the project's research plan that gives general outlines to the robot implementation and imposes several requirements on stakeholder and system level. The framework will be prototyped using the system features derived from the research plan. The research plan gives a structure to the process overall but project partners working on their parts of the system use their own domain specific design methods. This means that the actual design methodology and process is unknown, this will be considered when building the framework; the structure should be as process independent as possible.

Reasons given to SysML usage in the project is to address issues regarding design coverage, system integration and change management, but further usage is also possible as SysML can be used throughout an entire project life cycle. This adds a consideration of expandability in form of further usage. The created architecture framework should be structured so that it can be expanded. Even if extended usage would not be supported in the resulting framework itself, it is important that the resulting architecture model can be used in later project phases such as verification and validation activities.

As a multidisciplinary, multi-partner, EU-wide robotics project, Robominers is a suitable platform for a case study as it adds several considerations regarding the structure and features of an architecture framework. If these considerations can be satisfied and generalized it is assumed that answering these challenges with combined information from literature and findings from a case-study will yield satisfactory results when building a SysML based system architecture framework to be applicable to a wide range of mechatronic projects.

# 3. BACKGROUND ON SYSTEM ARCHITECTURE MODELING

Systems nowadays are rarely within one design domain, electronics and software are increasingly present in everything. As more design domains are included, complexity of a system also increases, this leads to the need of suitable design methodology and tools to manage the ever-increasing complexity. Systems engineering is multidisciplinary field of engineering, that focuses on designing and managing complex multi-domain systems and system of systems. In the words of Simon Ramo, systems engineering is *"...a branch of engineering which concentrates on the design and application of the whole as distinct from the parts, looking at a problem in its entirety, taking account of all the facets and all the variables and linking the social to the technological."* [5]

As systems engineering concentrates on managing the design process by looking at the big picture, its main focus is at the architectural level. “Architecture” is described by ISO/IEC/IEEE 42010 [6] as: *“fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”* and architecture description as “*work product used to express an architecture”.* In short, system architecture is the underlying structure of a system that can be documented as an architecture description. There are two situations that illustrate the difference: When architecture description is created prior building the system, the architecture description is an illustration or a vision of what the system will be like, the realized architecture could become different as the description can change during development. When created after building the system, the architecture description is an interpretation of the system that can vary depending who created the description. A description can fail to describe the actual architecture; hence it can be seen as a separate entity. Even though the terms are separated in the standard, they often have the same meaning in everyday language. Separating the terms does not always add value or understanding and can cause confusion. This thesis does not separate the terms: system architecture is used to mean architecture description.

Traditionally systems are defined by documents, this presents a problem in managing changes and tracing design decisions back to requirements across multiple documents. Model-based systems engineering (MBSE) is a system engineering methodology that uses a centralized system model in an attempt to shift away from documents. Centralized system models contain most of the relevant data regarding the system with inherent built in links between different system elements, this makes change managements and validation easier as many steps can be automated.

As noted earlier, architecture description can vary depending on who creates it. In order to reduce the variance and to get a comprehensive architectural model, guidelines should be used to ensure that all aspects of the system are covered, and well-defined modeling practices are used. Architecture frameworks are one form of these guidelines, ISO/IEC/IEEE 42010 [6] defines architecture framework as: *“conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders”.* A framework defines the elements and relations used to describe an architecture of a system of interest. Even though the term commonly refers to business or military related frameworks, they can also be used in systems engineering. In model-based systems engineering, architecture framework is somewhat essential in order to guarantee scope and cohesion of the model.

# 4. SYSTEMS MODELLING LANGUAGE (SYSML)

In a multi-disciplinary project, design teams use domain specific methods and notation to specify parts of the system. These notation methods may be unknown or not understood by the other teams, this leads to communication problems that can cause issues in the system integration phase if subsystems are found to be noncompatible. SysML offers high level architectural modeling notation, that combines all domains into one model, hence improving communication between the teams and creating deeper understanding of the system as a whole.

SysML is a general-purpose graphical notation language for systems engineering purposes capable of representing systems on an architectural level. It is used to model, analyze, validate, and specify complex multi-domain systems. SysML can represent system elements like personnel, hardware, software, and activities.

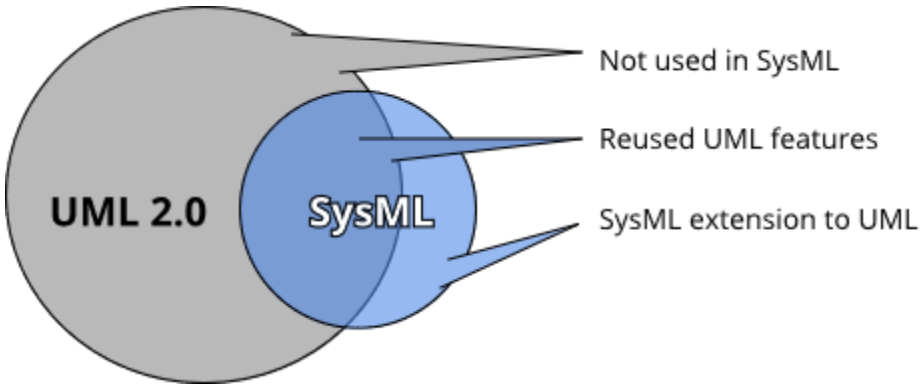


***Figure 1.*** *SysML / UML overlap*

SysML development started in 2003 as a request for proposal: “UML for Systems Engineering” issued by the Object Management Group (OMG) [7]. SysML is based on a subset of UML 2.0 with extensions to satisfy systems engineering needs (Figure 1). It was created by SysML Partners and is managed and published by OMG. As SysML is based on UML and the extensions are specified to be UML compatible, UML and SysML can be mixed and used in the same model, meaning that software architecture can be included in the system model.

## 4.1 SysML usage and tools

SysML enables modeling throughout the entire project lifecycle offering multiple usage modes and features that might not all be relevant in every situation. Because of this it is important to clarify its role and a purpose for its usage. Without a clear structure or plan, lot of the work put into modeling might be unnecessary or better used on elsewhere. SysML needs a methodology or guideline in order to be properly applied to a project, without it SysML is just a collection of diagrams without a purpose. A model created this way is like a “Mongolian Horde” [1]; large, lacking in structure and thus hard to manage. Therefore, establishing a model structure early in the process is vital. The methodology should be a MBSE process that is adapted to the project with SysML in mind. On the role of SysML it should be noted that it is not a design tool and should not be used as such. What SysML offers is a notation for representing a design, but the act of designing should be done with the tools suitable to the specific task. Most of the time a whiteboard is the best design tool. SysML is better used for documenting the design in a standardized form, providing a way to communicate the design to the people involved.

SysML Forum lists four levels of usage [8] from least rigorous to most rigorous. The first least rigorous usage mode *“SysML-as-Pretty-Pictures”* consists of using SysML notation to draw pictures without any of the more advanced features of SysML such as traceability or simulation. This type of usage does not utilize SysML to its full extent and results in a collection of diagrams that could have been drawn on PowerPoint. *“SysML-as-Model-Simulation”* is the second usage mode listed. This usage mode takes advantage of SysML’s simulation capabilities. SysML offers simulatable structures, but the simulation is dependent on a third-party simulation engine such as MATLAB/Simulink. While simulating some parts offers more than “pretty pictures” it does not create a system wide model. A more intended usage on SysML is *“SysML-as-Architecture-Blueprint”* that uses SysML to builds a system architecture model (SAM). SAM is a complete and precise representation of the system architecture and can be used as a “system architecture truth” to act as a reference for all involved stakeholders. Using SysML to define a system architecture is the level this work utilizes SysML. The next step would be *“SysML-as-Executable-System-Architecture”* that combines architecture and simulation by making the entire SAM simulatable and executable.

SysML is a notation language, by itself it does not do anything, it is just a language. Like all languages, it needs a medium, in this case a software tool. SysML is considered to be tool independent in the sense that it is not tied to a specific tool, but it is still dependent on a tool in order to be used. As a tool is needed, the tool itself plays a major role in the usage of SysML, different tools comply to SysML standard to a varying degree. Tools also might have some features, that are not required by SysML but augment its usability, for example traceability tools and interfaces with other software. There are multiple tools available ranging from free to professional. Project needs should be considered when choosing the right tool to use. Free SysML tools generally comply with SysML specification but may lack advanced features such as simulation or traceability. Free software might be sufficient for a small project, where some advanced features are not needed. As a project gets bigger and more complex, extra features might become essential.

## 4.2 SysML Diagrams with an example of a bottle

Following section goes over the different diagram types with an example of a bottle. The examples only serve to give a basic understanding of the different diagram types and do not show all SysML features associated to a specific diagram type.

SysML has 9 diagram types: Block Definition Diagram (**bdd**), Internal Block Diagram (**ibd**), Package Diagram (**pkg**), Parametric Diagram (**par**), Activity Diagram (**act**), State Machine Diagram (**stm**), Sequence Diagram (**sd**), Use Case Diagram (**uc**) and Requirement Diagram (**req**). The diagrams can be divided into static Structure Diagrams and active Behavior Diagrams (Figure 2). The structure diagrams are used to represent system structure and hierarchy between elements such as hardware, software, facilities, personnel etc. Behavior diagrams are used to describe the functionality and interactions of the system components.

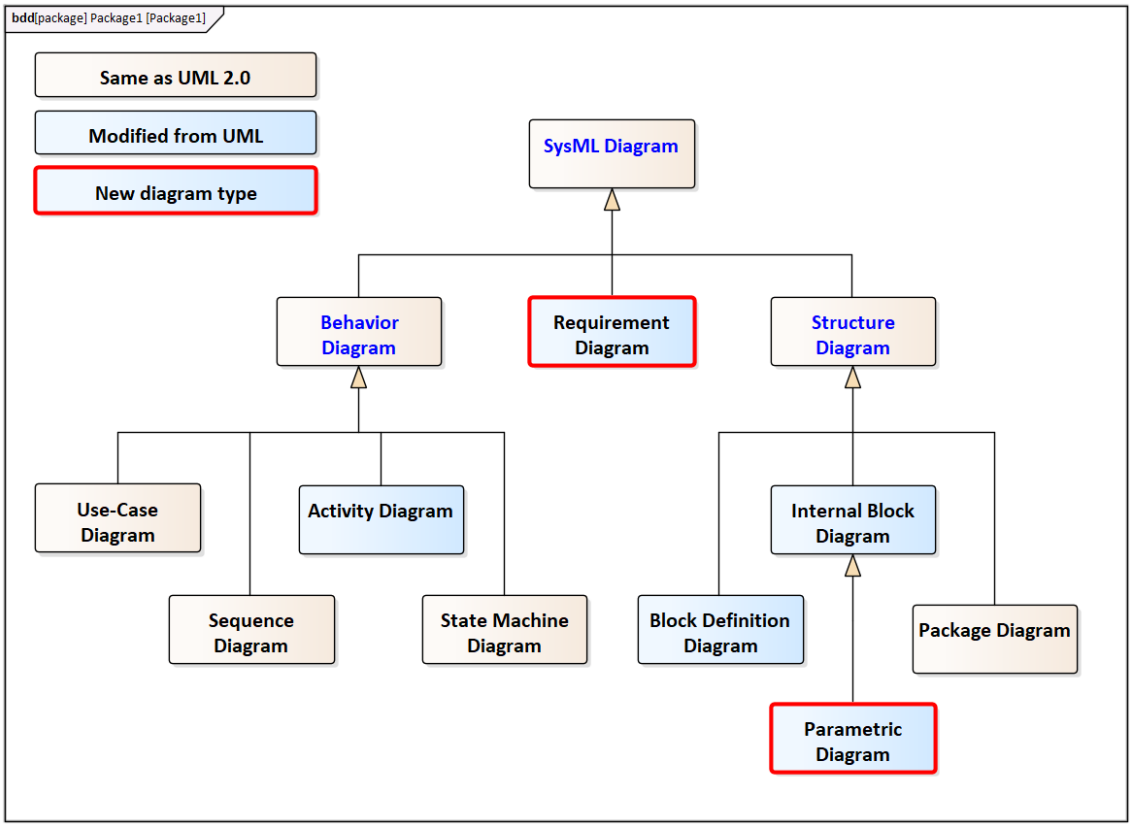


***Figure 2.*** *SysML diagram types*

Connectors and arrows in SysML point towards a higher abstraction level. This means that by following connectors in their direction the path ultimately leads to the highest-level objects that usually are the requirements.

### Block Definition Diagram

Block Definition Diagram is the most used and most versatile SysML diagram, it is used to define hierarchical composition of a system or a part. Common connector types are generalization and composition. Generalization is shown with a white-headed arrow and defines sub- and supertypes. Generalization used as the highest abstraction in the diagram can be used to add context to the element, for example in Figure 3 bottle is considered as a subtype of a container. Generalization as the lowest abstraction level can be used to show different implementations of the system element, for example bottle finish could be threaded or have a lip.

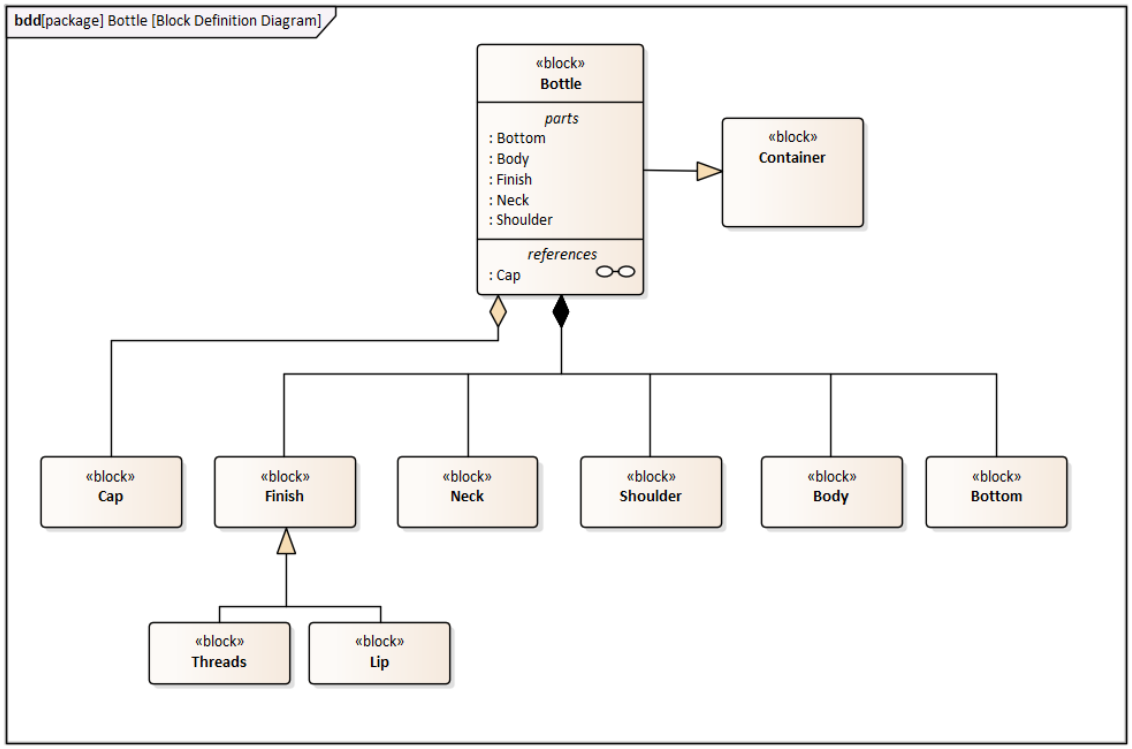


***Figure 3.*** *Block definition diagram: Bottle*

Composition has two types with a slight difference: black diamond arrow signifies a part composition and white diamond a shared composition. In the example above (Figure 3) Bottle has a cap with a shared composition and a bottom with a part composition. The difference is that a bottle can exist without a cap and still stay functional but becomes nonfunctional without a bottom, hence bottom is an integral part of a bottle, but a cap can be removed. Bottle is an easily understood example, but in more complex systems with multiple subsystems it might be harder to distinguish between part and shared composition. As the difference can be quite hard to perceive and the difference is small, the connectors can be used quite liberally, usually part composition is preferred to avoid confusion and shared composition is used with consideration.

### Internal Block Diagram

Internal Block Diagram uses the elements defined in a block definition diagram and shows the inner structure of a block. The physical structure of a bottle is presented in the Figure 4 bellow. Level of detail in the diagram depends on the purpose the diagram is used for, in this example most of the connections between the elements do not specify the type of connection. Only the connection between the cap and finish is presented in more detailed manner using ports

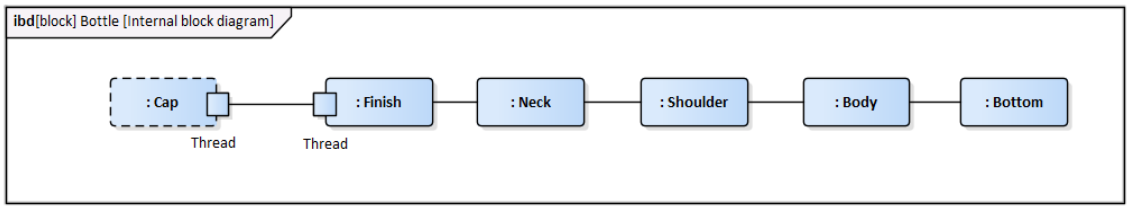


***Figure 4.*** *Internal block diagram: Bottle*

The cap was defined as a shared feature in the block definition diagram, this is reflected on an Internal Block Diagram as a dashed line on the cap element. The cap has a port named “Thread” and is connected via this port to a corresponding port on finish element. Connected ports do not require to be named similarly, for example “output” can be connected to “input”, alternatively the connector itself can be named without using ports.

### Package Diagram

Package Diagrams are used to organize the model. Diagrams and modeling elements reside in folders that in SysML are called packages. Package diagram are be used to visualize the structure of the model and to provide navigation links to relevant packages.

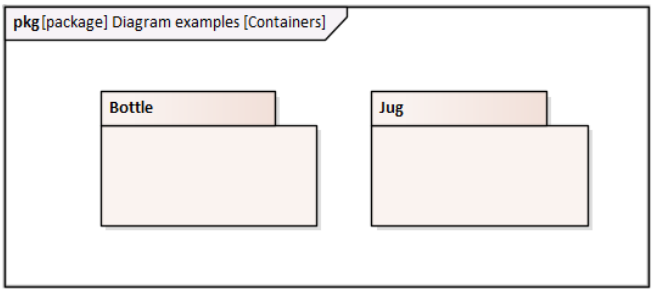


***Figure 5.*** *Package diagram: Bottle*

In the example (Figure 5), all modeling elements associated with Bottle reside in the Bottle package. Structure of the model is dependent on the used methodology and modeling guidelines, for example requirements and physical presentation could be in different packages.

### Activity Diagram

Activity Diagram is a behavioral diagram that pictures system behavior. The diagram, despite of its name, does not contain activities but rather actions that create an activity. Activity Diagram can in a sense be considered an internal diagram similar to an Internal Block Diagram as it shows the structure inside of an upper level element, in this case an activity block. SysML does not have a separate “activity definition diagram” for activities, but a block definition diagram can also be used to define activities.

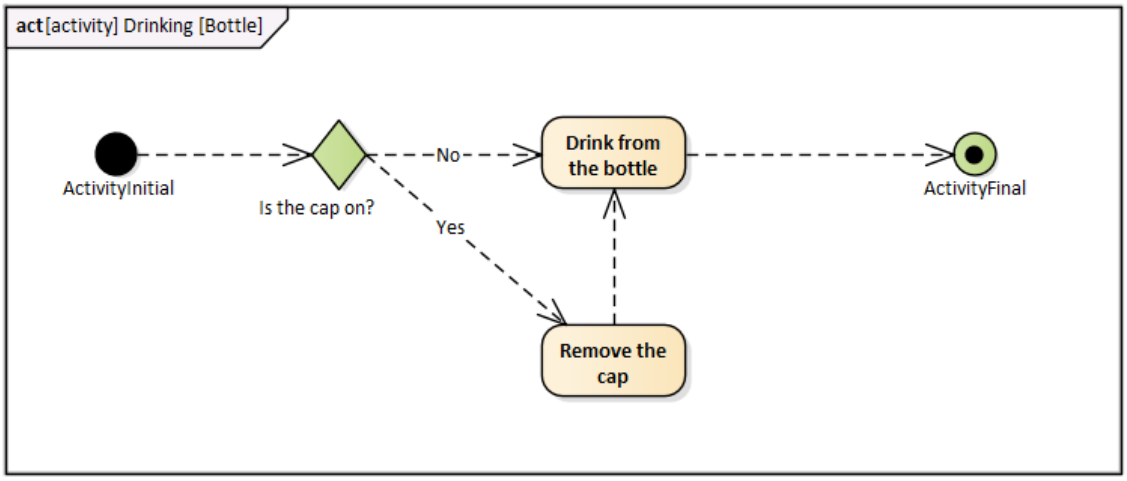


***Figure 6.*** *Activity diagram: Bottle*

Activity diagram shows a flow from start to an end of an activity with decision and junction nodes between actions. Figure 6 shows a simple example of an activity diagram for an activity named “Drinking”. Activity diagram can also depict a continuous activity without an end node.

**Requirement Diagram**

Requirement Diagrams are used to define, organize, and visualize requirements. The diagram forms a tree like hierarchical structure that shows relationships between requirements. The sub-requirements are connected to primary requirement with containment connector (circle with a cross). The sub-requirements can also be broken down to even further sub-sub-requirements when needed. New requirements can be derived by combining existing requirements.

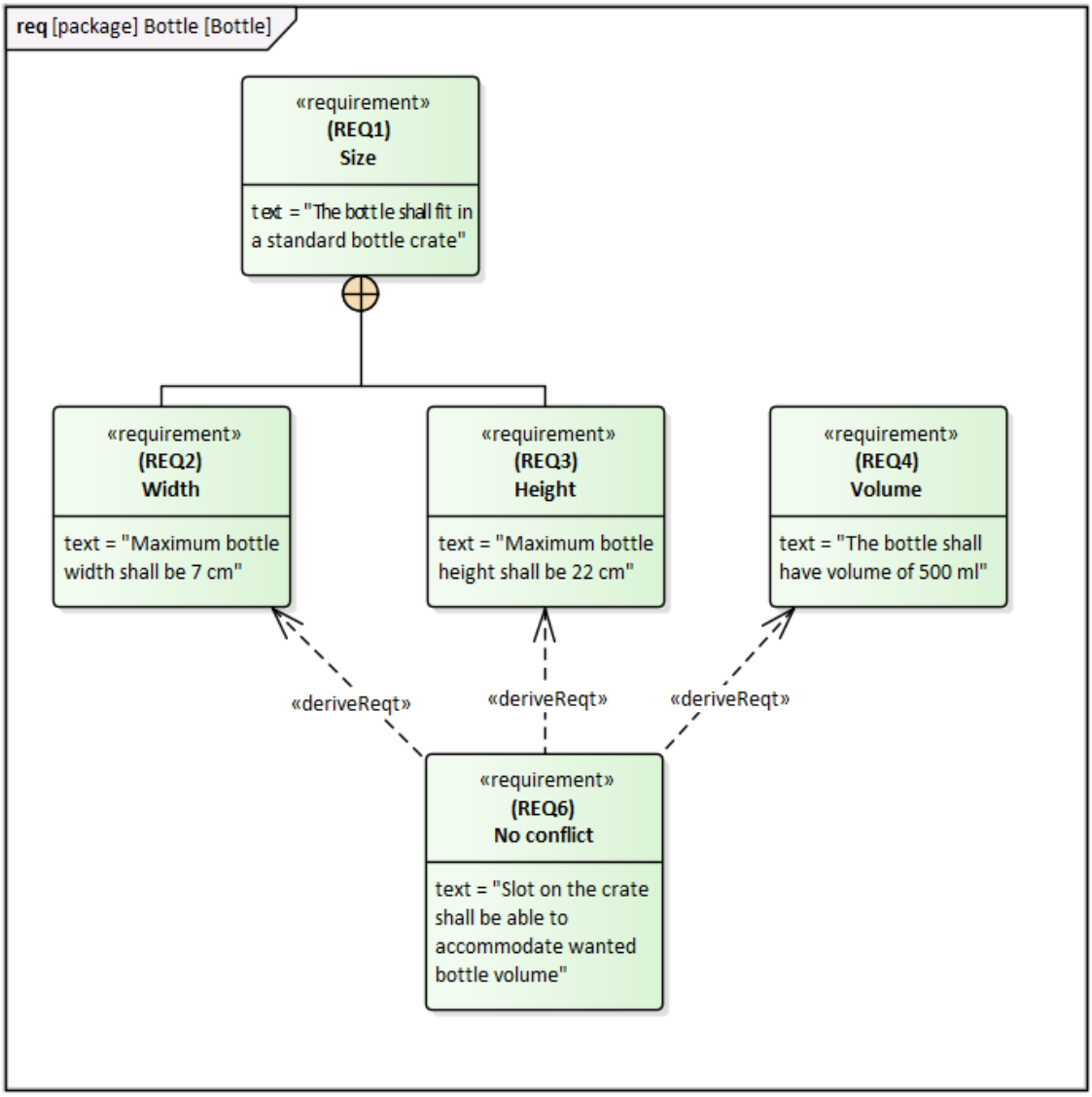


***Figure 7.*** *Requirement diagram: Bottle*

in Figure 7 there is requirement that “the bottle shall fit in a standard bottle crate” and requirement that “the bottle shall have volume of 500 ml”, when both requirements have to be satisfied, there is a potential conflict that has to be resolved. Requirement “Slot on the crate shall be able to accommodate wanted bottle volume” is derived from the conflicting requirements to be resolved.

### Parametric Diagram

Parametric Diagram is used to calculate and simulate different aspects of the system. The diagram consists of blocks that have can have constants, variables, or calculation formulas inside and ports as inputs and outputs. Connections carry the variables from block to block and the structure can be simulated to calculate the wanted results. Simulation requires simulation engine that is often offered as an extension to SysML modeling tools.

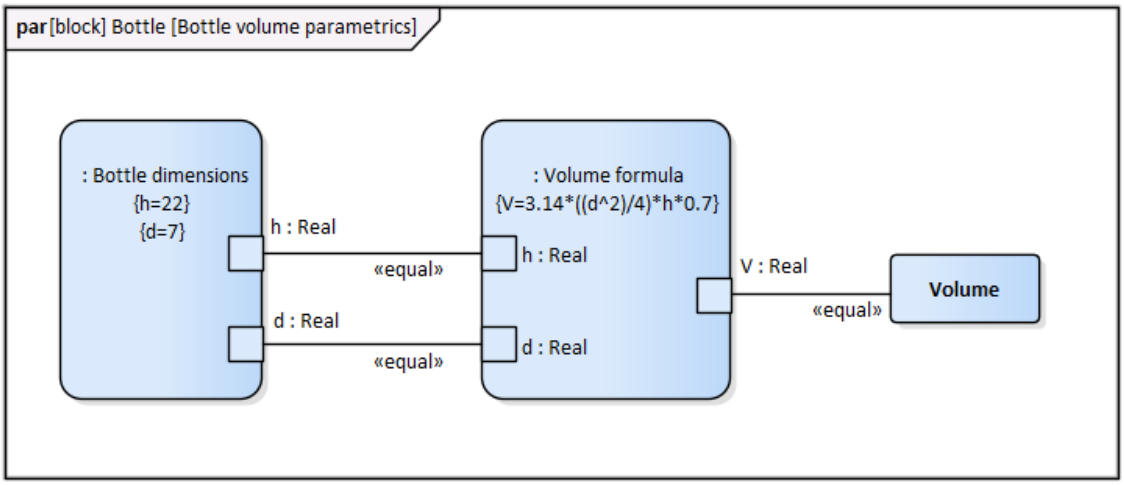


***Figure 8.*** *Parametric diagram: Bottle*

The Figure 8 shows parametric diagram for the volume of a bottle. Maximum bottle dimensions and required volume were defined in requirement diagram but it was still uncertain if the wanted volume fits the dimensions given. The diagram has two constraint blocks, one for bottle dimensions and one for volume formula, these blocks are linked together to output calculated volume. 500ml was required, when calculated the maximum volume that fits the dimensions is 592ml, this confirms that the requirements are not in conflict and can be met.

### Use Case Diagram

Use Case Diagram ties system actors to system functionalities, this gives and overview on how the system is used. Connections from actors to use cases represent communication paths between actors and the system. Actors can only communicate through the system and the only valid connection between actors is generalization that is used to define a supertype for an actor.

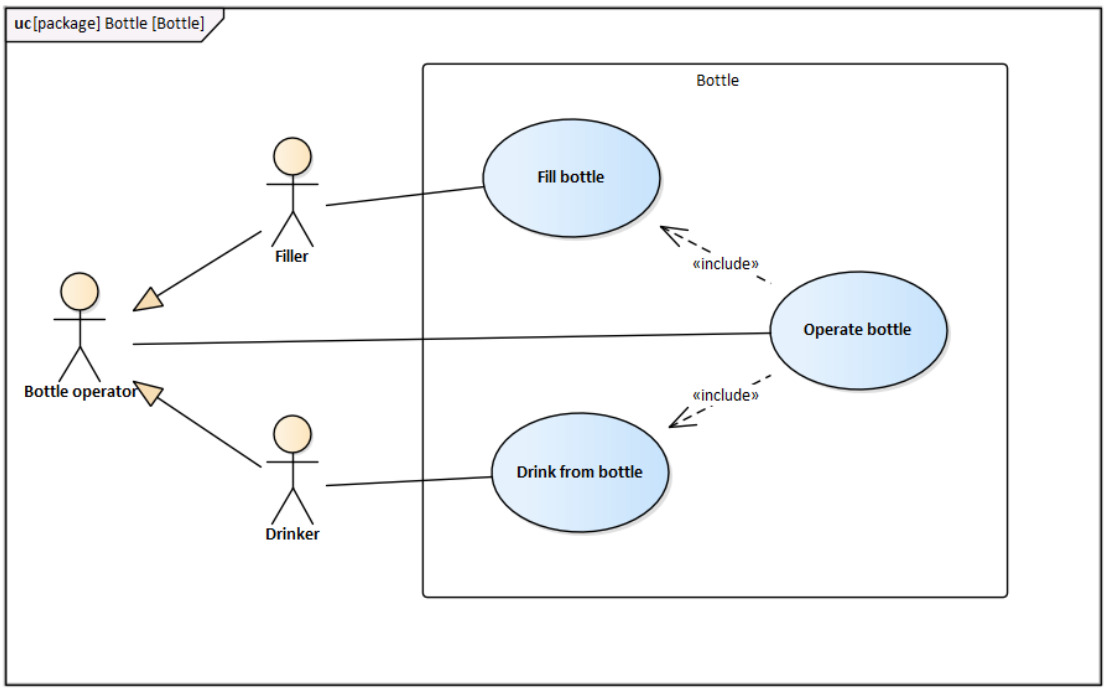


***Figure 9.*** *Use case diagram: Bottle*

Figure 9 defines two actors: Filler and Drinker that are both Bottle operators. The bottle has two use cases included in “Operate bottle”: “Fill bottle” and “Drink from bottle” that are associated with Filler and Drinker, respectively.

### Sequence Diagram

Sequence Diagram shows interactions between different entities within the system over time. System entities are shown as vertical lines with interaction messages between them. The diagram follows a single sequence of events that can be traced by following the arrows.

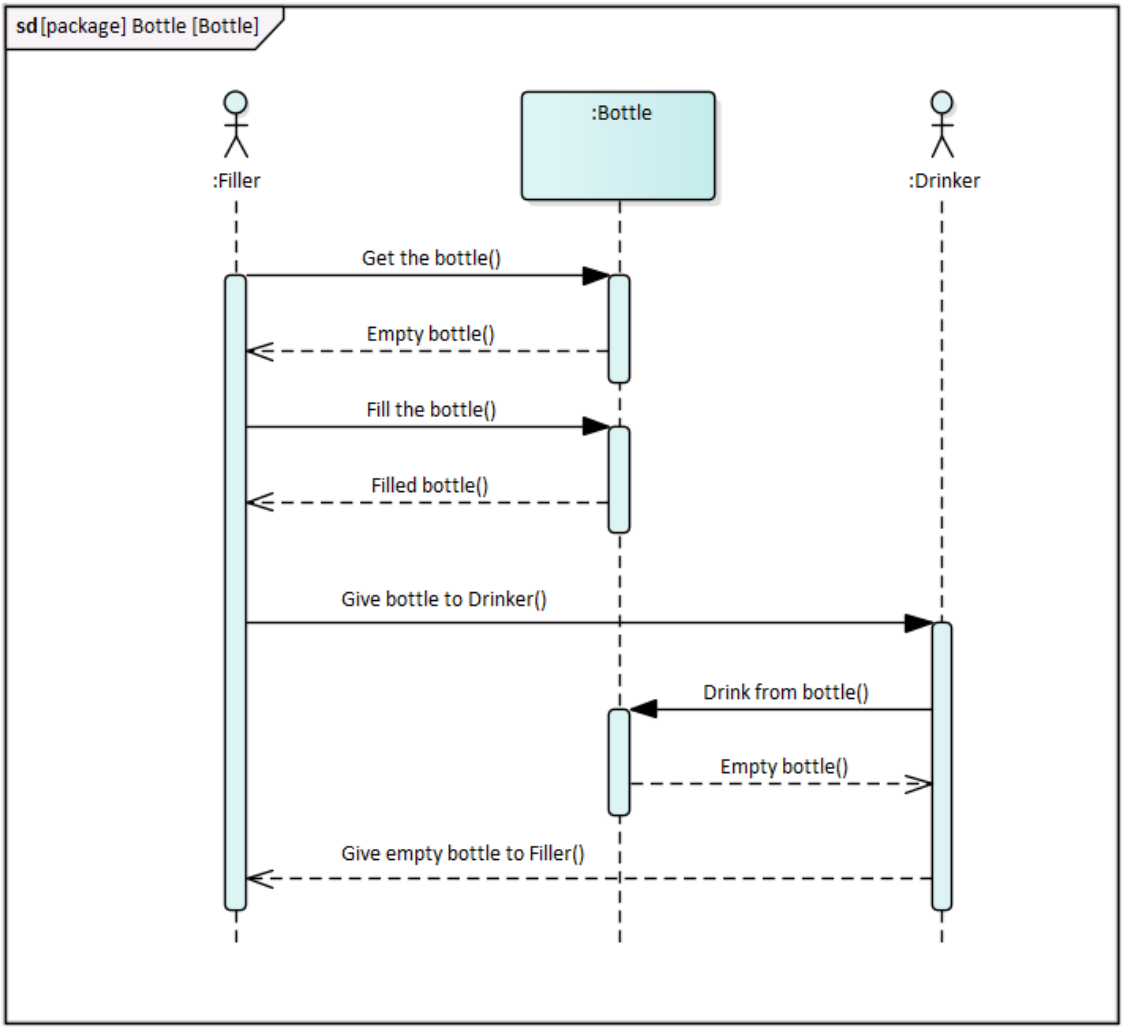


***Figure 10.*** *Sequence diagram: Bottle*

Figure 10 shows a scenario where a bottle is first filled and then dunk. This sequence has interactions between Filler, Drinker, and the bottle. Solid arrows are the primary messages and dashed lines are returns to the message. The sequence begins when Filler takes the bottle and gets an empty bottle as a return message. After the bottle has been filled, given to the Drinker and drunk, an empty bottle is again returned to the Filler.

### State Machine Diagram

State Machine Diagram is used to represent different states of the system with transition triggers between the states. These states can be passive or active, for example in Figure 11, the bottle has passive empty and full states and active filling and drinking states. Preferably these would be in separate diagrams.

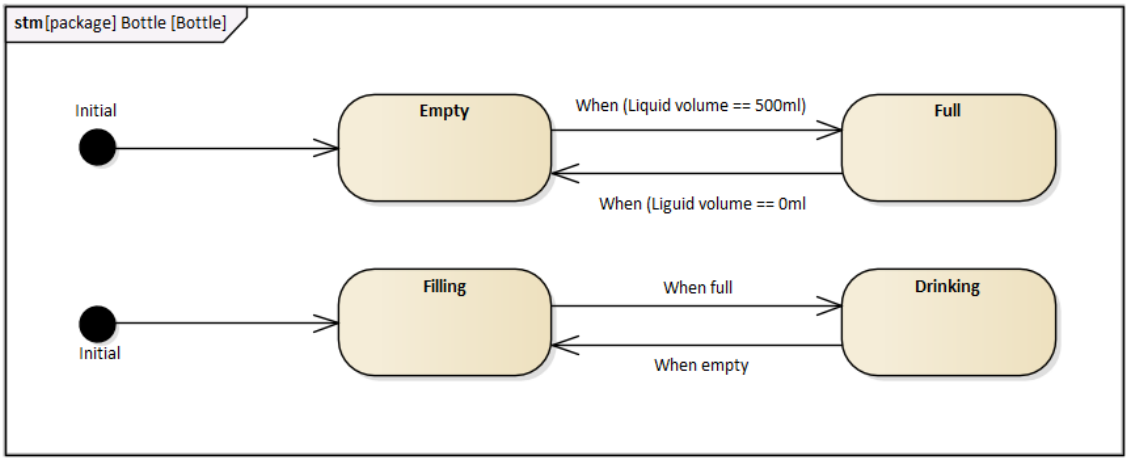


***Figure 11.*** *State machine diagram: Bottle*

### Cross-Cutting Constructs

The elements defined in their respective diagrams can be connected to elements in other diagrams. These connections form the foundation for traceability between different aspects of the system. For example, system elements that answer to a requirement are connected to requirements with a “satisfy” connector and parametric calculation can “verify” that a requirement has been met. These links do not have to be visualized on a diagram, they can be left in the background and inspected with a traceability tool.

## 4.3 SysML model structure

There are four levels to a SysML model that can be identified and distinguished from each-other: Underlying model, Visualized model, Package structure and Grouped structure. Underlying model includes all the modeling data, information of the modeling elements and connections between them. When something is modeled, it is included to the underlying model. Visualized model takes elements from the underlying model and shows them on a diagram. Usually elements are created on this level but when an element is removed from a diagram, it remains in the underlying model. This means that not everything has to be visualized at once; two or more diagrams can show different aspects of the same element structure, but on a different level of detail or focus on a different design domain. The modeling elements and diagrams are located in packages that form a folder-like Package structure that is commonly visualized in a "project browser" window (Figure 12). The structure is a collection of modeling elements and diagrams allocated to packages, when an element is created on a diagram, it is added to the model structure in the same package as the diagram. It is important to note that the visualized model does not have to follow the same form as the package structure. Elements form multiple packages can be used in one diagram.

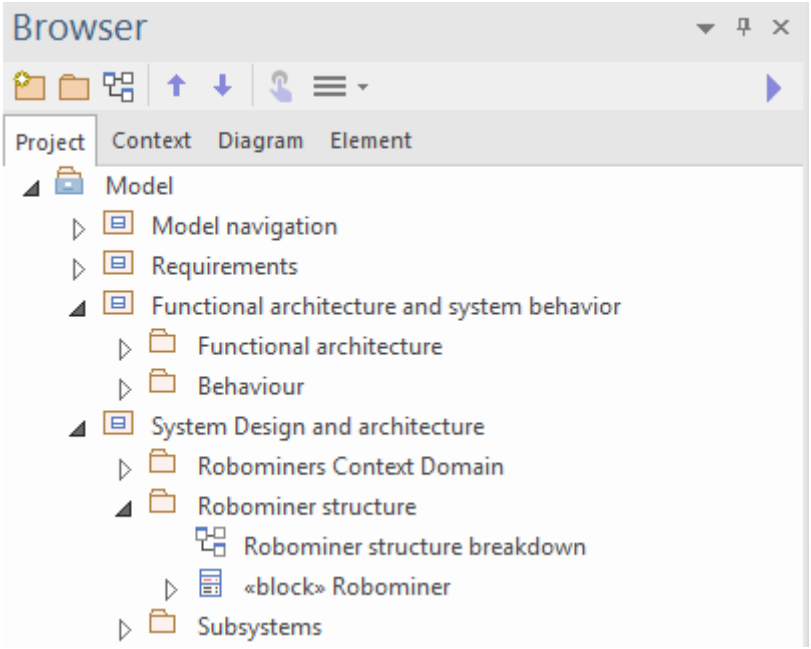


***Figure 12.*** *Model browser*

The model structure can also be grouped based on different needs. Grouped structure representation takes elements from the package structure and visualizes them with selected allocation. Grouping can be done with package diagrams or by using hyperlinks to selected diagrams. This can be useful when package structure is different from a design or process structure and visualization to those is needed.

# 5. APPLICABILITY OF SYSML IN MECHATRONICS DESIGN

Mechatronic systems consist of several subsystems from different engineering disciplines. These disciplines use their own jargon, tools, and notation to describe the system. The subsystems must be compatible in order to be integrated into one coherent system. For this to work, the different disciplines must be able to communicate and combine their design efficiently. Splitting the system into discipline specific parts is not always possible, and even then, interfaces must be defined in collaboration, design changes must be communicated between disciplines and the designs have to be integrated into one system. Common notation or integration of domain specific design tools might not be feasible nor desirable as some details would inevitably be lost. SysML offers a way to produce higher “system-level” models that combine domain specific designs on a discipline neutral abstraction level [9] acting as a communication medium and allowing different design disciplines to use their own methods internally.

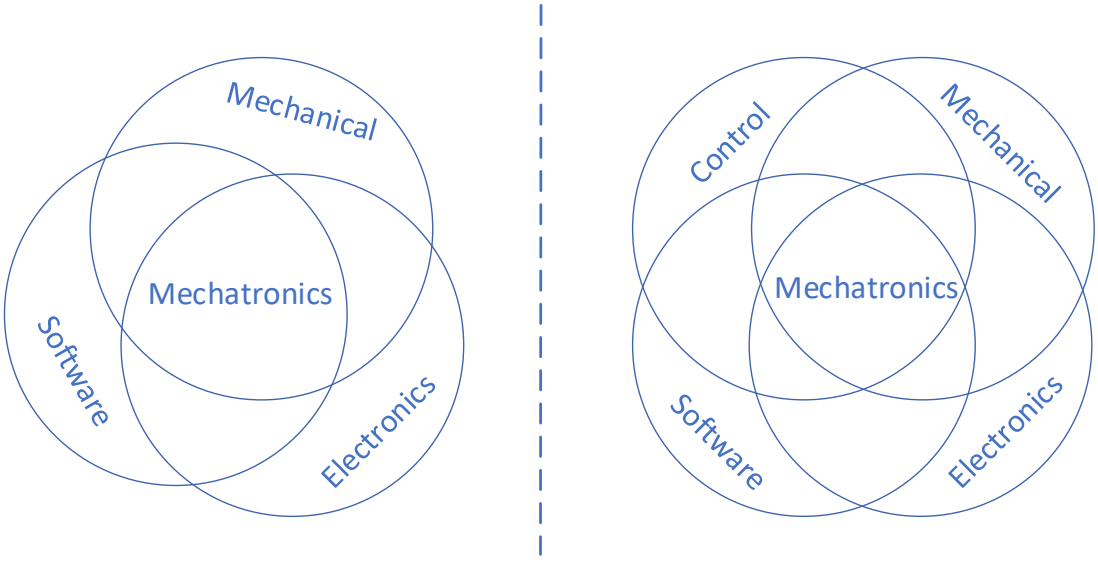


***Figure 13.*** *Disciplines in mechatronics*

This thesis considers mechatronics to consist of mechanics, electronics, and software. Control engineering is often also included into mechatronics but is left out here because it does not have the same presence in a system architecture as the other three disciplines. Control is still included in the architecture, not as components but as behavior between the components.

Figure 14 illustrates the transition from domain neutral architecture to domain specific design. The System Architecture Description is a discipline neutral system-level model that identifies the composition and logical structure of the system. This model can be created using SysML and the only discipline considered on this level is mechatronics that integrates all disciplines into one. This means that there is no distinction between the disciplines, and they can be combined into one model.

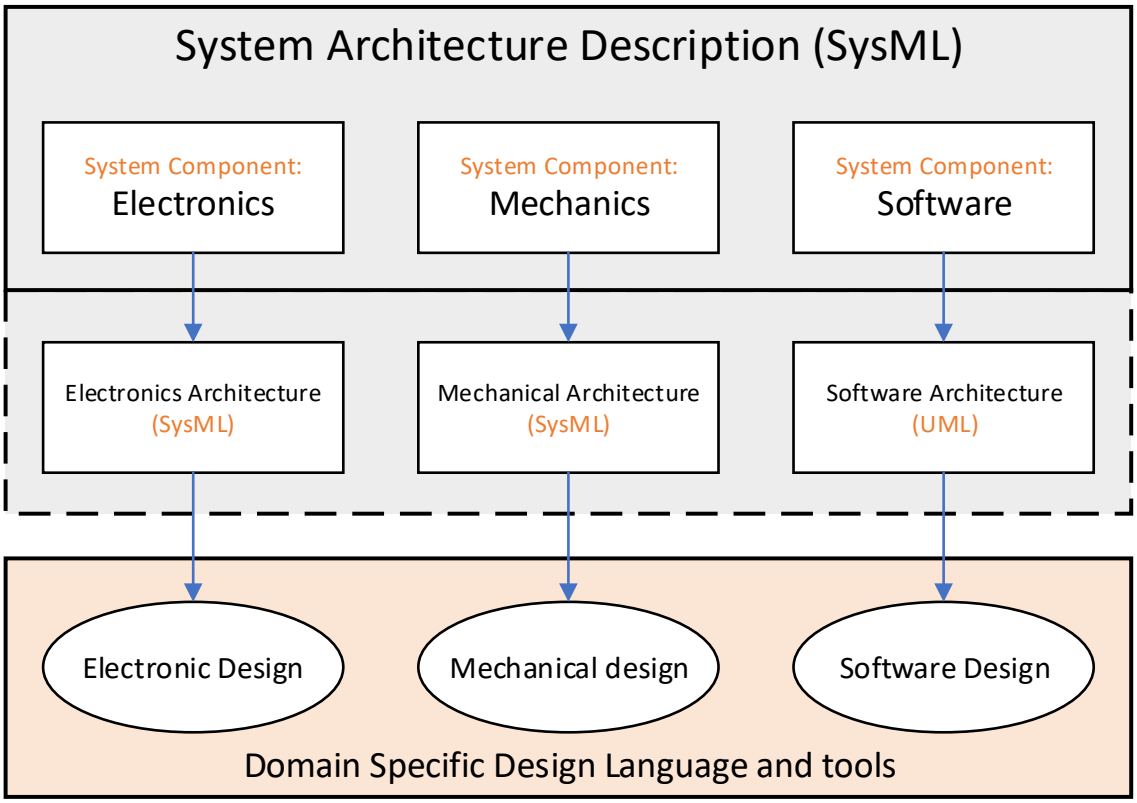


***Figure 14.*** *Transition from neutral to domain specific description*

The dashed line signifies the parts that are not strictly discipline neutral but are still part of the system architecture. The domain specific architectures can be included into the same SysML model as the System Architecture Description. Notable case is software architecture that can be created using UML. As SysML and UML are compatible, they can be used in the same model and links between elements can be established.

Analogues from spoken language can be drawn that illustrate how communication issues between disciplines can be solved and where SysML fits into this:

**Case1**: Proficiency in both languages. Two people could be proficient in both languages and change the language depending on the domain being discussed. This is unlikely to be the case in majority of mechatronics projects as people's expertise is usually concentrated on one domain.

**Case2**: Mixed language. Two people do not fully speak each other's language but know parts of it. When speaking they keep thing simple and do not use difficult phrases. Hand sighs and pictures are used. In a multidomain workplace this would be like coffee room discussion between colleges, sharing information about their fields without going to specifics.

**Case3**: Interpreter. Two people speak through a third person who interprets what the other one is saying and translates to another language. In mechatronics this could be using a manager or a system expert to relay information between disciplines.

**Case4**: Common language. Two people speak a common language, that might not be the first language on either. This is the way SysML solves communication issues, by giving engineers a common language.

## 5.1 Benefits and drawbacks with SysML in mechatronics

SysML can be used in a MBSE process as a part of the overall modeling methodology. It has been noted that most methodologies don't have an organized modeling framework for SysML to guide the actual modeling and thus are too general to be adopted as is to a project [10]. Many methods can be adapted to work with SysML such as SYSMOD but might require custom profiles that might not work on all platforms [11]. When a methodology is adapted to support SysML, a modeling framework and guidelines should be created to organize the model and guide the modeling efforts. One of the main barriers of adopting SysML as a part of a MBSE process is the lack of these frameworks and guidelines and high initial learning effort [2] [3]. When applied on a project SysML offers a centralized source of information and can act as an architectural reference to design teams working on a project. With SysML as a common language, design teams from different disciplines can get a deeper understanding of the system, helping the design process. In mechatronics, there are dependencies between subsystems from different engineering disciplines and there can be multiple disciplines working on the same subsystem, so communication on a lower system level is also required. Whereas lower levels are useful for the communication between design teams, the higher level can be used to get a broader picture of the system and manage the design process. Communication with stakeholders can also be done on a higher level as stakeholders' interests usually lie in what the system is capable of not how it does it.

Several studies have found SysML too general to fully support mechatronic design. Multiple workarounds have been proposed that usually involve usage of customized extension profiles or tool specific features to add support to satisfy specific needs. Ozkaya [12] has identified several shortcomings regarding SysML requirement diagrams, some of them can affect other diagram types as well. Firstly, diagrams can become extensively cluttered when one to many and many to one relations are present in the diagram. This results in reduced legibility of the diagrams as readability and possible mistakes become a problem. Ozkaya is also concerned about the abstraction gap between the diagram and the actual design, that is present as the diagrams are always an abstraction of the design and cannot always capture it fully. SysML can in some ways be too general to capture some aspects of a design. To support these use cases, additional properties should be added to the SysML blocks.

G. Barbieri et. al [13] propose usage of an extension profile, that alleviates some of the shortcomings brought up by Ozkaya. Properties can be added to SysML blocks by using extension profiles, that can be used to redefine certain aspects of the language or add new features. Their model defines specific blocks for every abstraction level, this helps to break the model down to abstraction levels and to distinguish the elements on different levels. Abstraction levels can also reduce the complexity of a diagram as only one level is presented at a time. To further help with the readability it is recommended that requirement diagrams are presented in a tabular format.

Criticism has also been made [14] that SysML does not have a proper model-based requirements definition structure. While the requirements are in the model, the requirements themselves are not model based. Requirements still have a static definition that is not linked to anything. A true model-based requirement would have a definition that changes automatically to accommodate a change that affects it. With current SysML implementation, requirements have to be tracked and modified manually when a change happens.

Requirements are not the only aspect that has been found lacking. SysML does not have specific elements to represent system functionality, this has led to recycling of other elements and diagram types to define system functions. Lamm and Weilkiens [15] propose usage of activity blocks on a block definition diagram to define a “function tree” to act as hierarchical representation of system functionality and usage of internal block diagram to define functional architecture.

Whereas most studies focus on providing solutions and extensions to certain parts of SysML, more extensive profiles have also been created that heavily redefine SysML to support interdisciplinary design. SysML4Mechatronics [16] defines domain specific blocks and brings more UML features to SysML. Even though extension profiles do alleviate issues caused by lack of features, they have their own issues. Implementation of custom profiles can be heavily tool dependent and usage of custom profiles can cause confusion in a multiuser scenario as they need familiarization on top of already complex SysML.

## 5.2 Requirements modeling and management

Requirements play an important role in guaranteeing success of a project. Jon Holt et al. [17] gives reasons to why understanding requirements is important. Firstly, requirements drive the project, everything in the project should be connected to the requirements to ensure that the project results in the wanted outcome. Requirements also increase confidence in design decisions. When decisions are based on requirements, they can be justified, requirements act as a communication tool with stakeholders and end-users, to ensure satisfaction.

SysML takes a model-based approach to requirements engineering. Requirements are modeled as requirement blocks and connected into a hierarchical tree structure. The requirements are elicited from a requirement source, that can be included in the SysML model as a block or an artifact. A source is not the requirement itself but raw information that the requirement is extracted from. Source can for example be standards, laws, requirements given by a stakeholder or user stories. Most requirement descriptions are an interpretation of the underlying need, meaning that they might not be accurate. Jon Holt et al. [17] emphasize that all requirements should have a source documentation they can be traced back to. When traceability is established in the model, requirement validation process becomes easier and potential errors can be found.

Ozkaya [12] has identified some issues that the default SysML requirement diagram has when mechatronics is concerned. When drawing a hierarchical structure and tracing the requirements to their respective source elements, the diagram can become cluttered with connection lines. One way to solve this is to only give the source for the primary requirement, this would work if the secondary requirements all has the same source, but often that is not the case. Another problem that is faced with requirement diagrams in mechatronics is that SysML has very generic requirement elements. Default elements lack certain fields that would be useful in when defining requirements for mechatronic systems. Barbieri et al. proposes a customized SysML requirements profile to be used [13]. The profile would have additional requirement types to define specific element for each hierarchical level. These types would add more suitable properties to the requirement elements such as classification, category, and related module / sub-system.

A requirement model might become hard to perceive as its complexity increases. Therefore, it is important to note that SysML requirements diagram is not a requirements documentation, hence the requirements should be exported into Excel in a tabular format to allow easier management. Furthermore, the modeling process should be methodological. Development processes often starts by gathering, analyzing, and specifying system requirements and use these requirements to drive the process. SysML has features, that cover requirements modeling, but without a rigorous methodology or framework, the requirement model coverage might remain lacking.

Jon Holt et al. have presented an approach to context-based requirements engineering (ACRE) in their book Model-Based Requirements Engineering [17]. The approach is based on multiple views that show different aspects of the requirements. These views and / or their contents are linked to each other to form the ACRE framework. Traceability between elements is regarded essential to the requirements engineering process. Tracing things by hand is tedious and error prone, model-based solutions are recommended, SysML in particular. Some of the notable ACRE views include:

Source Element View listing all requirement sources so that they can be linked to the actual requirements that are elicited from them.

Requirement Description View showing requirements and their properties such as name, ID, priority, and text.

Requirement Context View showing requirements in a context and looks at them at a certain point of view. Use case diagram is recommended to show functional requirements in relation to the realizing system element.

Context Definition View showing different contexts, that could be used in the Requirement Context View. Context can for example be a stakeholder perspective, subsystem, or a level of system hierarchy.

In their framework J. Holt et. al. underlines the importance of traceability and deeper understanding of requirements. With the views they establish a well-defined structure that attempts to ensure the completeness and correctness of the requirements model while setting it up for further use in later process steps. When traceability links are maintained in the model, they can be used for verification and validation activities throughout the entire project life cycle.

## 5.3 System functions and architecture modeling

System architecture defines the overall structure of the system and identifies the sub-systems and their relations that must be considered in the design process. Architecture model can be used to capture design decisions and when linked with a requirements model, it can be used to validate the design. It is important to note that SysML is not a design tool, it is better suited for modeling and analyzing the designed system. That said, SysML can be used to support the design process acting as a design reference to manage complexity and to find conflicts in the design. System architecture model considers all aspect of the system equally in a discipline neutral manner showing the composition of the system and relationships between its parts. This way an architectural model acts as a reference or a blueprint for further design.

The link from requirements to system architecture is recommended to be done through a functional architecture [18] [19] [15], that specifies the system functions and the interactions between them. The reason behind the use of functional architecture is an aspiration to create a design that is not tied to a specific solutions or technology, but to higher-level concepts that transcend technological limitations, allowing model reusability. This might not always be possible, as often, before the design process even starts, some decisions have already been made that specify the technology that will be used. There are two ways to link functional architecture to physical architecture. Functions can be allocated to realizing system elements or the system elements can be allocated to the functions they serve. G. Barbieri et al. [13] allocates functions to system elements as attributes, if one to one mapping from functions to system elements can be established, this approach works well as it would make the system architecture easy to read as functions would be cleanly under system elements. However, there are times when multiple elements are needed to realize one function, this leads to a situation where the elements end up with the same function as an attribute, this in turn can cause confusion in understanding how the function is realized. One way to alleviate this problem is to group the functions that need multiple elements to realize into one bigger entity and allocate that to a sub-system.

Even if no issues were to arise from using functions as attributes to system elements, this would still cause inconsistencies in traceability as functional architecture is a more abstract concept than a physical architecture. From traceability point of view connector directions should be kept facing towards a higher abstraction to allow fluent analysis of the system. This in mind system elements should be allocated to functions. This is the approach proposed by Lamm & Weilkiens [15], they show that by first creating a functional architecture, and designing system architecture based on it, results in deeper understanding of the designed system and a model that can be updated to use different technological solutions supporting multiple generations of a product.

Functions do not have a dedicated element in SysML, nor is there a specific diagram type for them. Lamm & Weilkiens define a functional decomposition using Block Definition Diagrams with Activity Blocks instead of basic Blocks that are commonly associated with the diagram type. The Activity Block elements are then arranged into a functional architecture in form of an Internal Block Diagram. Lamm & Weilkiens use the functional architecture to identify what sub-systems are needed to realize the functions, similarly F. Mhenni et. al. [19] allocates the functions based on their perceived generalized component class such as “motor” or “distance sensor”. This identification is done with a Block Definition Diagram that creates a hierarchical presentation of the components involved in the system. When the “building blocks” have been identified, the architecture can be formulated using these components. There are many ways an architecture can be designed, SysML only provides the notation language for it. The resulting architecture and model reflect the used methodology and process.

A system architecture can be modeled on different abstraction levels. G. Barbieri et al. [13] divide the system into multiple levels from System level (highest) to component level (lowest) and apply certain views to all elements on every level. Views include Requirements View, Design View, Integrated view, and Behavioral View. These views are used in the same manner as J. Holt et al. [17] use views in their ACRE framework. While the ACRE framework focuses only on the requirements engineering, combining it with the approach from G. Barbieri et al. it can be generalized to cover the whole architecture.

## 5.4 Further model usage

SysML can be used to document and model different system features throughout the whole product design lifecycle and supports all major phases of the process. After the model has been created there are several ways it can be used during and after the design process.

**Architecture reference**

The most common use of a system architecture model is to use it as a reference for further design. As a high-level model, it combines all design disciplines onto one presentation of the system, allowing different design teams to use it to collaborate further design decisions. When more specific design is included in the model, it acts as a document that can be referred to during integration and when planning manufacturing. The model can also be used to communicate with the stakeholders, giving them an overlook of the system on a relevant detail level showing them how the system relates to their interests.

### Change Impact analysis

Design process is often iterative, and changes may have to be made at some point. These changes to the design usually effect other parts of the system as well. SysML model elements are linked and traceable to each-others, this makes impact analysis easier as the model automates some parts of the analysis. Some tools have specific features to support impact analysis, but even without those tools, SysML models inherently make analysis easier. Changes can be traced down and upstream depending on where the change is made, when a requirement changes during development, its impact has to be traced downstream, there are some approaches that attempt to automate this process [20], some SysML tool also provide tools that create an impact chain for a specified element through the established links in the model. When a change is made at a lover system level, its impact might not impact upstream elements, but the change still has to be validated.

### Verification and validation

R. Baduel et al. [21] show that verification and validation (V&V) activities are present in every phase of the design life-cycle and should have well defined set of rules at the start of a project. Verification rules define how modeling elements are created and what properties they should have and act as a guideline for modeling. The model is verified based on these rules, verification confirms that the system is modeled correctly, but does not confirm that the model itself is correct. Verification, in a sense, acts as a “grammar check”, validation checks for the actual content of the model. R. Baduel et al. propose that verification is done as part of the modeling activity and a completed model or a part is validated by a domain expert. Validation activities are based on system requirements, validation rules define different aspects of the system that must be validated against requirements. J. Holt et al. [17] have similar approach regarding system requirements that can be generalized to cover the whole system. They define a set of rules that have to be followed during modeling activities and use diagrams to validate that the requirements can be implemented.

# 6. CASE-STUDY: AUTONOMOUS ROBOT DEVELOPMENT

The case-study is done on Robominer research project that aims to develop a bio-inspired robot capable of autonomous underground mining, this is a complex multidisciplinary task that will require close communication between different design domains and multiple teams on different sites. The preliminary phases of the project are used as a basis for this case-study. Findings from the literature study are applied and adapted to suit the project needs. Robominers project proposal and research plan are used as reference material to create, prototype, and refine a modeling framework for design of autonomous robotic systems.

## 6.1 Background and purpose of the case study

The Robominers project adopts a model-based approach using SysML. The project has multiple partners working on their parts of the robotic system using their own domain specific processes. Project management has proposed SysML to be used during the project to address issues regarding system integration, design coverage and change management. A model-based approach can address the aforementioned issues by combining the design into one model so that it can be analyzed as a single entity with subsystems connected to each other with traceable links.

Some of the primary aspects of SysML are documentation and traceability, that allows easy tracking of design decisions for further development or for a next iteration of the product. As Robominers is a research project, after it is finished, there most likely will not be next iteration. There are concerns that SysML might just add unnecessary work and burden to the project. For SysML to be worth using, the benefits must outweigh the added work its usage causes. While documentation for further development might not be useful for the project, it does enable traceability and analysis capabilities during the development. This adds a consideration to the SysML implementation. The modeling process should be planned so that it does not take up considerable amount of resources. At the same time, the model should be structured so that certain aspects of the system are grouped together to enable easy analysis of the model, so that the issues regarding integration, coverage and change management can be addressed.

Modeling needs a well-defined structure in order to leverage full benefits of a model-based approach. Without a clear structure the model may become unmanageable in size and composition, meaning that it becomes exceedingly hard to validate the correctness of the model. It is also noted that modeling guidelines are crucial for the modeling team members to be able to work on the same model without communication or consistency issues [21]. The purpose of a framework is to create a well-defined modeling structure and to keep the model well managed allowing cooperation between modeling teams while ensuring a proper coverage and cohesion of the model.

The case-study is done during the starting months of the Robominers project as a preliminary preparation for the design process. Goal of the case-study is to apply literature findings and recommendations to generate modeling data that can be analyzed to derive an architecture framework that can and be generalized for use on wider range of projects. The thing that is not widely discussed in literature is the actual implementation of SysML into the general project workflow. The structural elements and usage of specific parts of SysML are usually the main focus but creating an easy to use modeling structure seems to be often overlooked. The model easily becomes hard to navigate when system complexity increases. This complexity has to be managed in order to preserve model integrity and cohesion and increase the fluidity of the workflow. The case-study tries to answer the question of how SysML model should be structured to fluently fit into a workflow. The resulting framework should aid the SysML modeling process by providing a clear structure and tools to make the process fluent and precise.

Even though a research process differs from a commercial design process in more ways than the importance of documentation, it is assumed that this case-study is a good platform to develop and test a SysML based system architecture framework that can be reapplied to another projects. In order to accommodate different design processes, the framework is built to be process independent.

## 6.2 Framework design process

The process starts by modeling several aspects of the system, Figure 15 provides a rough outline for the process. The system is first modeled with recommendations found in literature but without a modeling framework, this means that the first draft will not have a clear coherent structure. The framework is then formulated by organizing and rearranging the model into a logical structure that can then be reapplied to another modeling process. The process has three parts: modeling the system, rearranging the elements into relevant groups, and finally creating a framework based on that grouping.

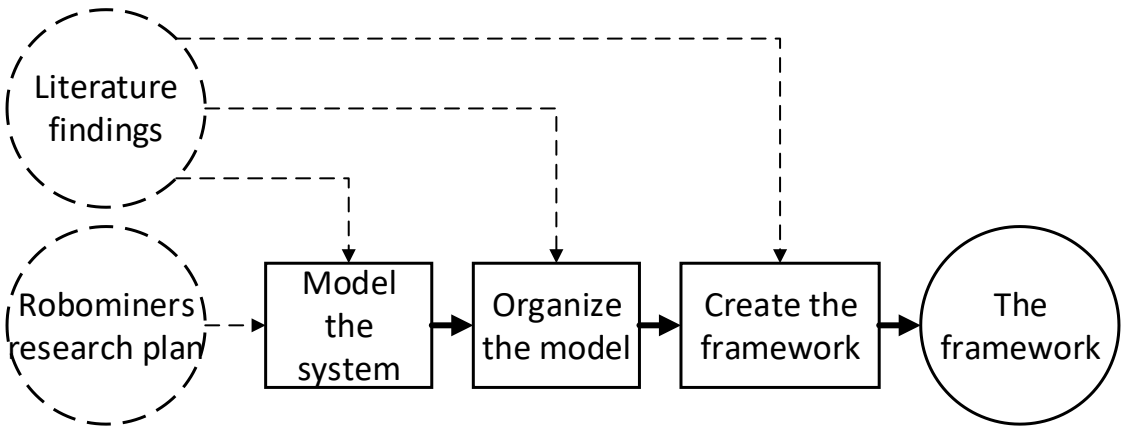


***Figure 15.*** *Framework design process outline*

To start the process, several aspects of the system were modeled without any concrete structure and it was confirmed that the model indeed does become a “Mongolian horde” [1]. At this point the elements and relations were present, but the model was hard to navigate and manage. The next step was to organize the model and try different element groupings. SysML preserves elements and links independently from their usage in diagrams, this means that established elements can be rearranged into multiple diagrams without having to redefine them.

Two major approaches to organize the model were identified during modeling: Element centric and process centric, each having their own advantages and disadvantages. The two approaches differ in how they threat modeling elements in the modeling process.

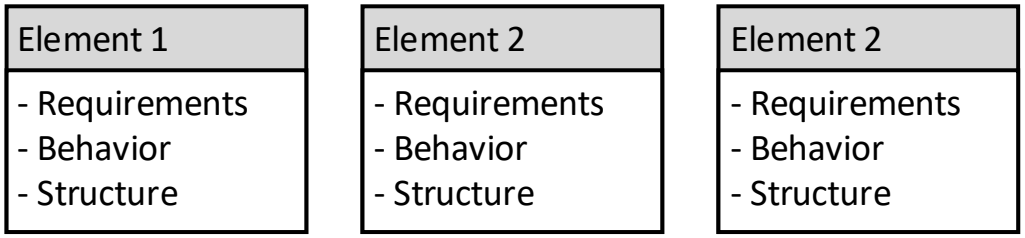


***Figure 16.*** *Element centric approach*

When the model is organized based on the system elements (Figure 16), each element has their own properties under them in the model structure. This means that it is easy to analyze a single element as all its properties are easy to find. The downside to this approach is that the modeling sections (requirements, behavior, functions) become fragmented. This means that during modeling these sections cannot be addressed as a whole. This leads to a structure that works well on smaller scale projects or on a very general level but might not scale up to accommodate more complex systems where it is important to be able to address bigger entities.

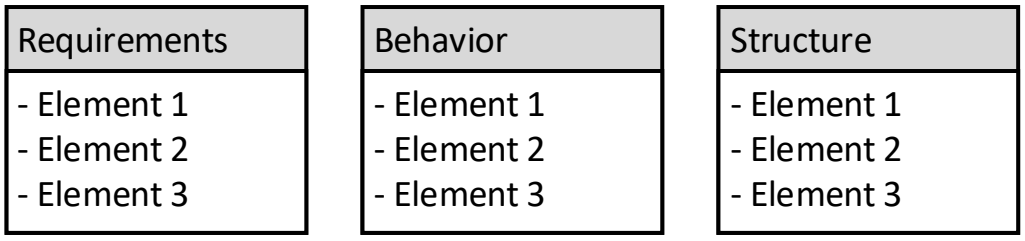


***Figure 17.*** *Process centric approach*

Process centric approach (Figure 17) organizes the model based on the different modeling steps and sections, all element properties of the same type are located in one package. This makes the modeling process easier as all properties related to a specific modeling process are present in the same place.

Modeling was divided into three distinct parts that were kept separate: requirements, system functionality and system structure. The process is visualized in Figure 18. After examples of all three aspects were modeled, it was studied how the information could be combined into one framework that shows the relations and importance of all three aspects. To help with navigating the model, additional navigation package was created to provide hyperlinks to different parts of the model. Navigation links also form the basis for the visualization of the architecture framework and are utilized in other modeling structures as well [10] [22].

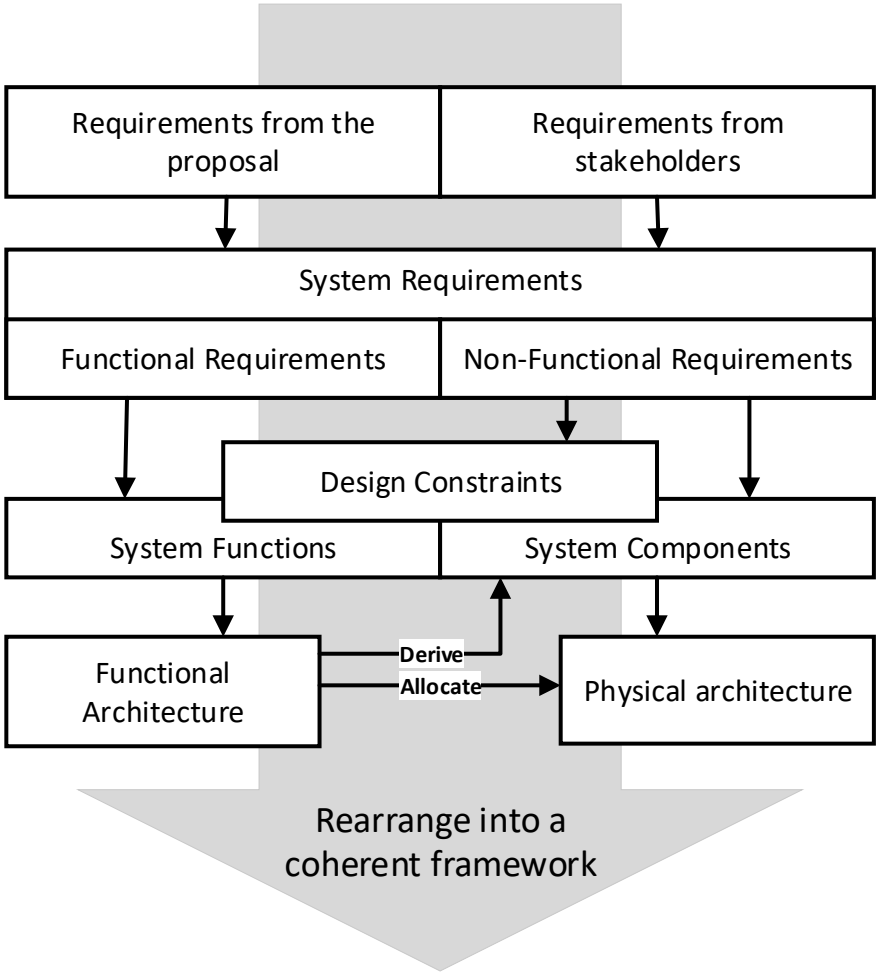


***Figure 18.*** *Modeling process*

## 6.3 Requirement modeling process

Requirements were the starting point of the case-study. The first step was to identify the requirement sources e.g. stakeholders that should be considered in the project. Different stakeholders give requirements on differing importance levels; Requirements from law and standards are not negotiable, management and engineering might impose some restrictions on what is possible, and end-users have different requirements depending on their role. It is important that all stakeholders related to the system of interest are identified and considered in order to avoid conflicts later in the development. A general stakeholder structure provided by J. Holt et. al. [17] was used to accomplish this (Figure 19). The diagram categorizes and lists possible roles and provides a template for identifying stakeholders. Elements in the diagram act as folders that contain source material for the requirements in an unedited raw form.

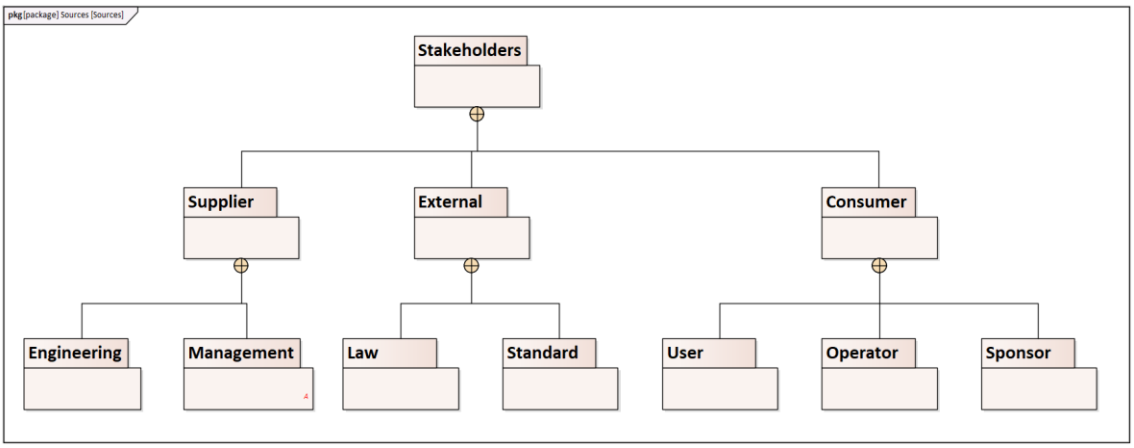


***Figure 19.*** *Source tree*

The first set of requirements comes from the research plan that outlines the research subject and some specified parts of the system structure, for simplicity, this was the only source used in this case-study. As the research plan is a ~70-page document it had to be divided into smaller section in order to be processed efficiently. These sections were placed into the SysML model as blocks that enclose html formatted text that can also include pictures. These blocks act as the source elements that the first set of requirements is elicited from.

The diagram defining the source elements is very simple and the main purpose is that the source elements can be linked to requirements. Source diagrams are located on the packages defined in the source tree. In Figure 20 the Robominers project proposal / research plan has been divided into sections. These sections are small enough that when a requirement is elicited from the source, it is easy to see where the requirement has come from. Small text snippets can be combined into a section / chapter using containers, this helps readability of the diagram.

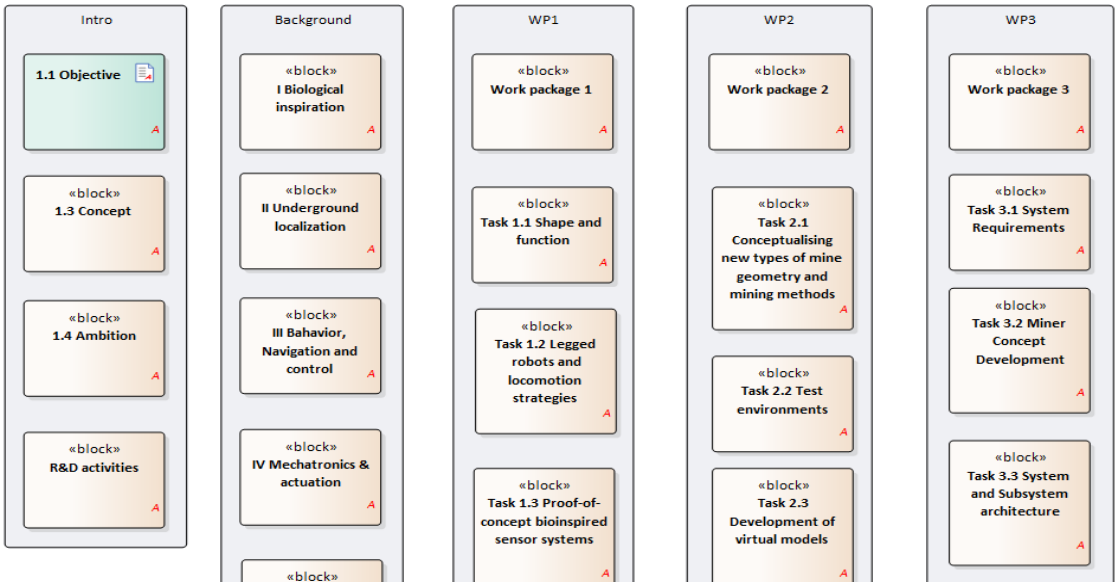


***Figure 20.*** *Source elements*

Source elements are be modeled as blocks that enclose the source documentation, but other elements can also be used if found suitable, It might be a good idea to model the sources as document artifacts as blocks are also used to model system elements and that might cause confusion. Before eliciting the requirements from the source material, it was decided to categorize the requirements based on their type (Figure 21). This was done in order to keep individual diagrams from growing too large. The categorization was done based on the design domain a requirement would be associated with. This resulted the categories being: Mechanical, electric, software and mechatronics.

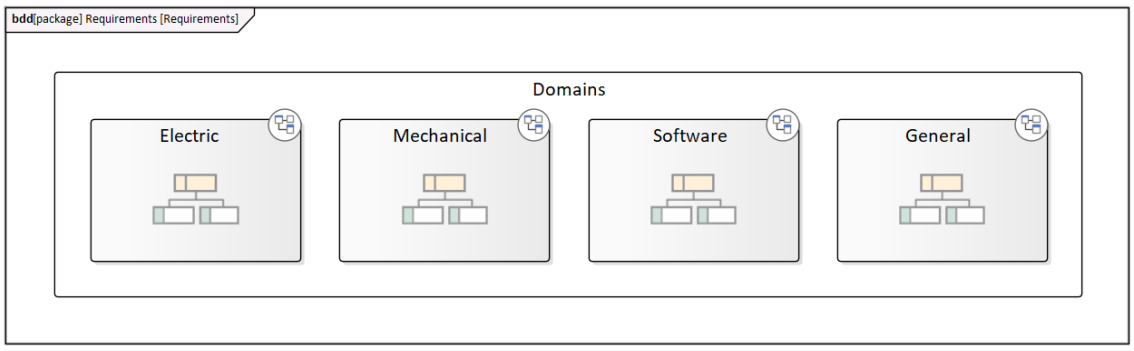


***Figure 21.*** *Requirement domains*

Requirement diagram was created under each domain, to house the requirements elicited from the source material. Diagrams form a hierarchical tree structure, that starts from a source and fans out as sub requirements and derived requirements, there can be multiple trees in one diagram. In the example bellow (Figure 22) the tree starts by deriving requirements from earlier ones. Requirements are modeled as SysML requirement blocks; however, default requirement profile does not have all the properties that are needed to specify requirements properly. In the Figure 22 the blocks used are system requirement blocks that are defined with a custom profile that adds more properties to the default requirement block.

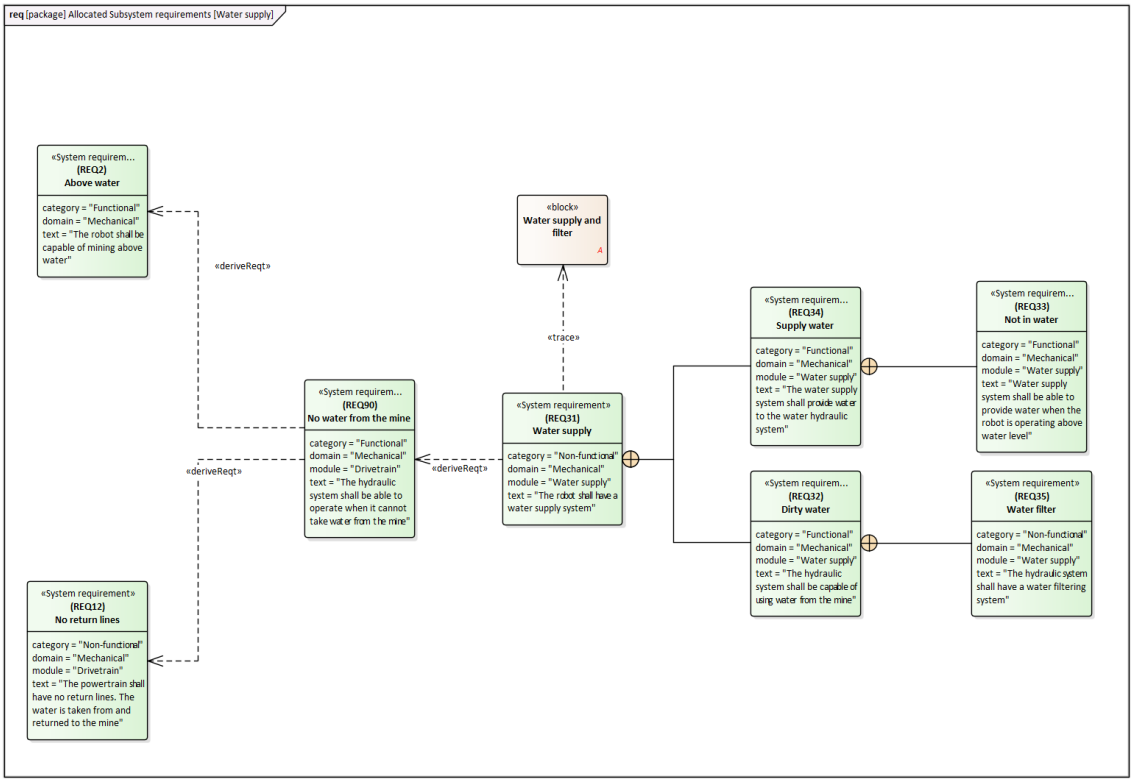


***Figure 22.*** *Requirement diagram*

When a requirement is linked to a traced source that source element does not have to be drawn on the diagram, it can be hidden and remain in the background. This makes the requirement diagram easier to read. Source elements were drawn only for the primary requirements and if a secondary requirement had different source, only then that was drawn. Secondary requirements with the same source as the primary requirement would have their sources hidden. Satisfying system elements can also be included into the requirements diagram if needed, this would however add complexity to the diagram and reduce readability, so it is recommended to keep those connections hidden. On the other hand, requirements diagrams are more of a modeling tool than a design reference, so these diagrams are not looked at directly, but rather act as a basis for traceability. To use the requirements on a design process, the requirements should be exported to a more formal document. The modeling tool used (Enterprise Architect) supports csv-formatted exports to that was used to transfer the requirements to a spreadsheet.

Requirements from the SysML diagrams should be documented in tabular format (Figure 23) in order to enhance usability. Exporting the requirements from the model to a spreadsheet gives a more usable representation of the requirements. On a spreadsheet, the requirements are more accessible and can be easily arranged based on different tags.

| | A | B | C | D | E | |
|---|---|---|---|---|---|---|
| 1 | Category | Domain | Type | ID | Name | Text |
| 2 | Functional | Mechanical | System requirement | REQ32 | Dirty water | The hydraulic system shall be capable of using wate |
| 3 | Non-functional | Mechanical | System requirement | REQ35 | Water filter | The hydraulic system shall have a water filtering sy |
| 4 | Non-functional | Mechanical | System requirement | REQ6 | Bio-inspired | The robot shall be bio inspired |
| 5 | Functional | Mechanical | System requirement | REQ1 | Amphibious | The robot shall be capable of mining underground, |
| 6 | Functional | Mechanical | System requirement | REQ4 | Underground | The robot shall be capable of mining underground |
| 7 | Functional | Mechanical | System requirement | REQ5 | Underwater | The robot shall be capable of mining underwater |
| 8 | Functional | Mechanical | System requirement | REQ3 | In slurries | The robot shall be capable of mining in slurries |
| 9 | Functional | Mechanical | System requirement | REQ2 | Above water | The robot shall be capable of mining above water |
| 10 | Non-functional | Mechanical | System requirement | REQ7 | Legged locomotion | The robot shall use legs for locomotion |
| 11 | Non-functional | Mechanical | System requirement | REQ13 | Open loop | The powertrain of the robot shall be based on an o |
| 12 | Non-functional | Mechanical | System requirement | REQ15 | Water hydraulics | The robot shall use water as a pressure medium to |
| 13 | Non-functional | Mechanical | System requirement | REQ12 | No return lines | The powertrain shall have no return lines. The wate |
| 14 | Non-functional | Mechanical | System requirement | REQ31 | Water supply | The robot shall have a water supply system |
| 15 | | Mechanical | System requirement | REQ16 | Mechanical connection | The multicoupling shall connect modules machanic |
| 16 | Functional | Mechanical | System requirement | REQ90 | No water from the mine | The hydraulic system shall be able to operate wher |
| 17 | | Mechanical | System requirement | REQ8 | Electrical connection | The multicoupling shall have an electrical connecto |

***Figure 23.*** *Requirement table*

The benefit of using SysML to model requirements rather than going straight to a table is that in a table the requirements are detached from the system and cannot be easily traced up- and downstream. With SysML, the requirements are part of the model and linked to the rest of the system. This allows traceability that can be used to resolve conflicts between requirements and track where and how a certain requirement has been satisfied. In some projects, requirements might be handled outside of SysML with another tool such as Rational DOORS. In these cases, the modeling tool might have integration with the requirement management tool, but most tools also support csv-formatted imports that can be used to bring requirements into the model. Without a close integration, requirement ID can be used to link the requirements on an external tool to the requirements in the SysML model.

## 6.4 Function and architecture modeling process

A robotic system can have several subsystems with different functionalities and dependencies. These features should be addressed as they play an important role in defining the system architecture. As recommended by Lamm & Weilkiens [15] system functions are described with a functional architecture, that identifies the individual functions and their relations to other functions. First step is to identify the individual functions, requirements specify some of the required functions but not all of them. Requirements generally specify what tasks the system must be able to accomplish. Tasks are usually composed of several smaller tasks that the system must also be capable of in order to accomplish the original task. Tasks can also have prerequisite tasks as well that might not be mentioned in the requirements but nevertheless must still be accounted for. Furthermore, tasks generally do not only have one way to carry them out, there can be several, some harder and some easier to realize.

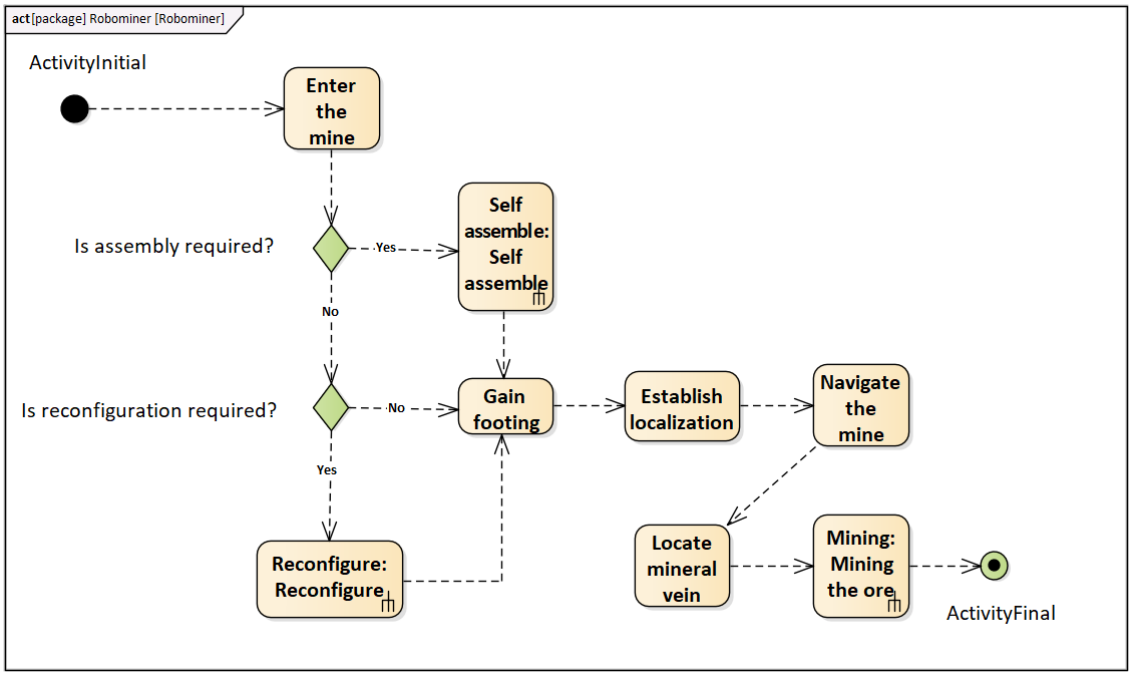


***Figure 24.*** *Behavior modeling*

Without modeling the required task, its sub-tasks, prerequisites, and structure, it can be hard to see what concrete functions and parts the system must have in order to be able to accomplish the task. Activities and functions were modeled in parallel, using each other and requirements as a reference. Figure 24 shows an activity of entering the mine and finding an ore vein to mine, this identifies several functions that the system is required to have in order to accomplish the modeled activity. The activity was modeled based on the system requirements and the mission statement, that also do specify some of the functions in the diagram. The purpose of modeling behavior was also to find possible gaps in the requirements.

The identified functions are documented as activity blocks, on a Block Definition Diagram forming a hierarchical tree structure (Figure 25).

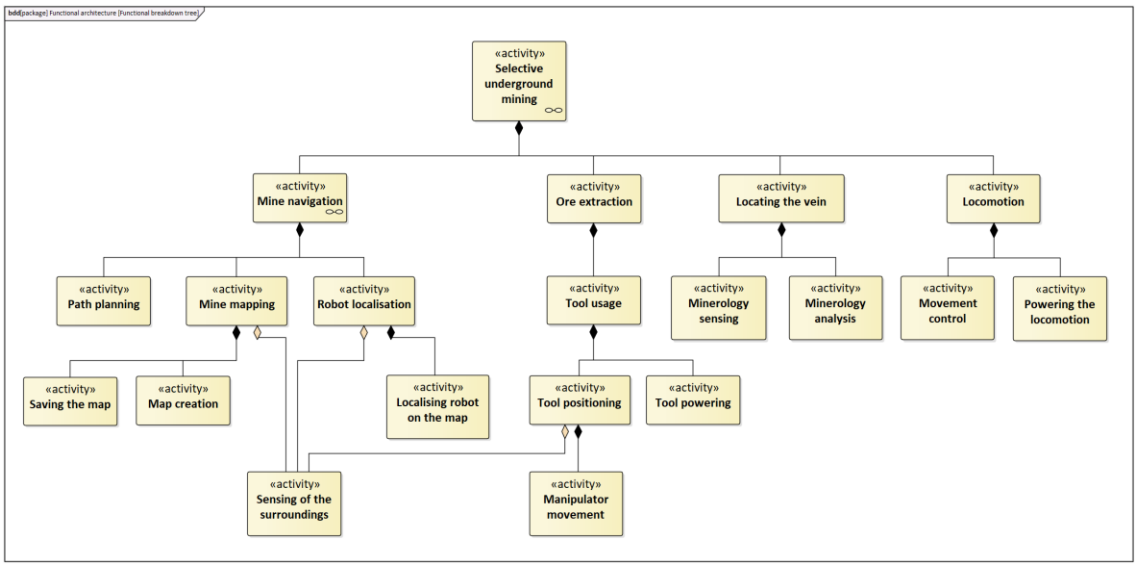


***Figure 25.*** *Identified functions as activity blocks*

First high-level functions were identified and divided into sub-functions, this results in a hierarchical tree structure. Identification of functions started from the main function of the system that was “Autonomous selective underground mining” that was also one of the modeled activities. This primary function was divided into smaller functions based on the system requirements and behavior modeling. Next step was to form a functional architecture that shows the functions in relation to each other specifying the inputs and outputs of each function. The first layer under the primary function were used to form the highest level of the functional architecture.

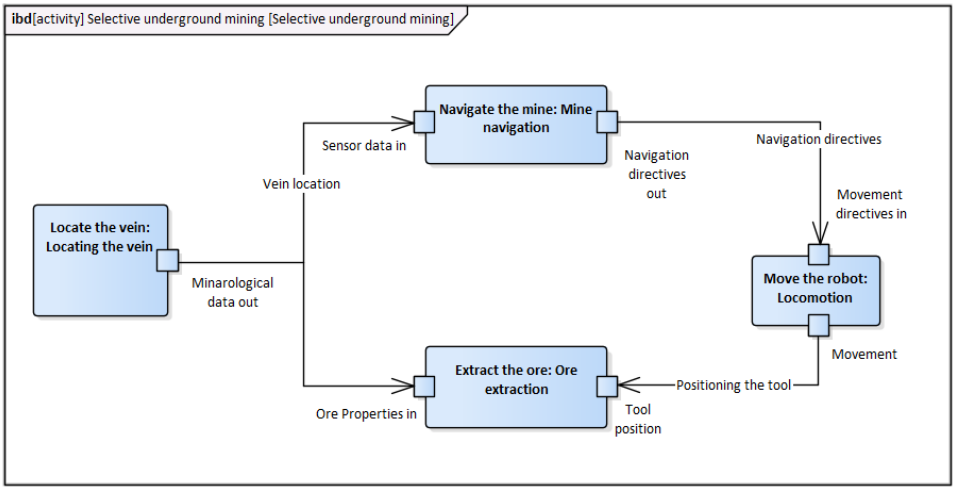


***Figure 26.*** *High-level functional architecture*

The functions on the higher-level architecture were broken down into a more specific functional architecture that uses lower level functions that form the higher-level function. Example in Figure 27 shows “Mine navigation” function and identifies that it requires additional input “High level directives in” that should be included and resolved in the higher level model.

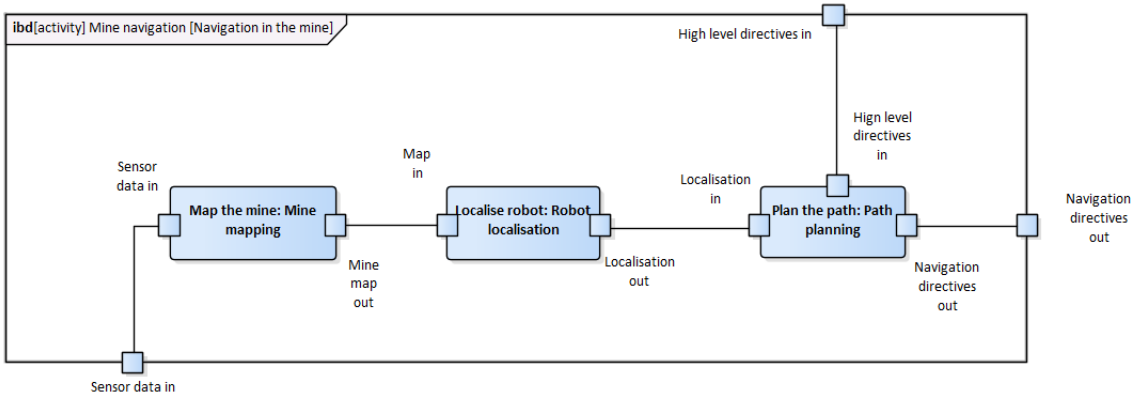


***Figure 27.*** *Lower level functional architecture*

When all higher-level functions have been modeled in more detail, the model forms a network of functions connected through inputs and outputs that can be used to identify what sub-systems and components the system needs in order to realize the functional architecture. There can be several possible implementations of architecture and it is up to the designer(s) to choose the most suitable one. The actual design and the design process were not paid much attention to as the focus of this study is to explore modeling techniques and combine them into a framework. Figure 28 shows the identified sub-systems and defines the system composition. The figure also includes some auxiliary sub-systems that were not part of the functional architecture such as “Power system” that supplies electricity to the rest of the sub-systems. Auxiliary systems are somewhat dependent on the chosen system architecture and technology so in order to consider their place in the architecture in a more formal manner, their behavior should be modeled in order to form an individual functional architecture for them.

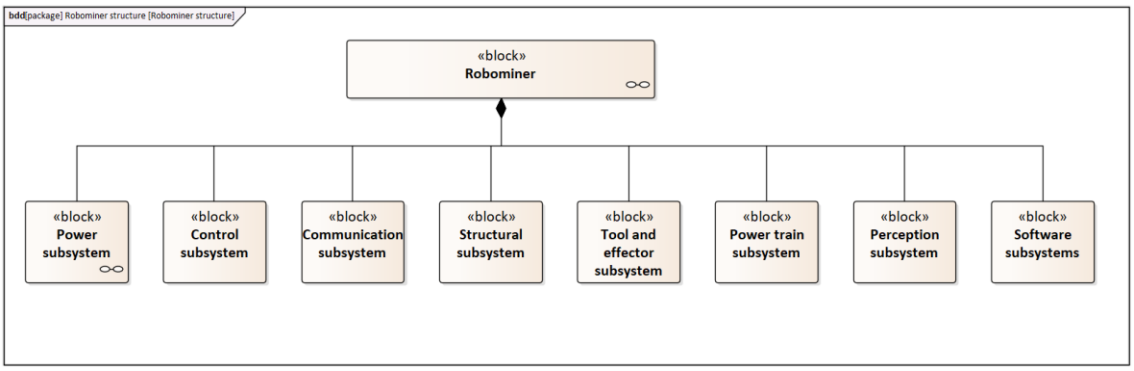


***Figure 28.*** *Decomposition into sub-systems*

After the system composition was defined, a high-level architecture was modeled using Internal Block Diagram (Figure 29). The high-level architecture does not have well defined interfaces between the sub-systems as they are not yet known. At this level, the model serves as a rough outline that will be refined in later modeling steps. The connectors between the sub-systems can be named and ports can be used if there is understanding of their type.

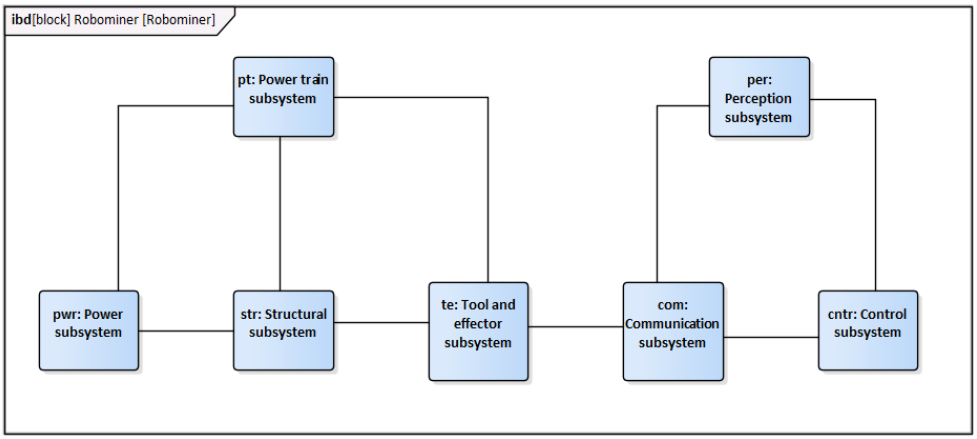


***Figure 29.*** *High-level architecture*

The same process was repeated for the subsystems defined in the system composition, first identifying the components needed and documenting them on a Block Definition Diagram (Figure 30). The functional architecture was also used as a reference on this level to identify what parts the sub-system needs to realize the functions.

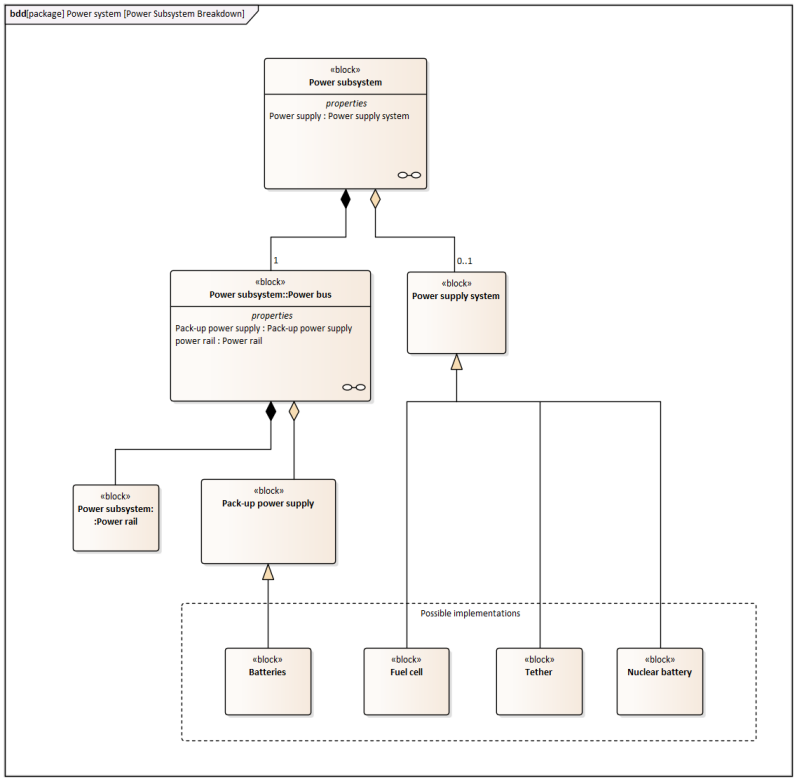


***Figure 30.*** *Sub-system composition*

More formal interfaces were created for sub systems while modeling their internal structure (Figure 31). When the interfaces are known, they can be updated to the higher-level model.

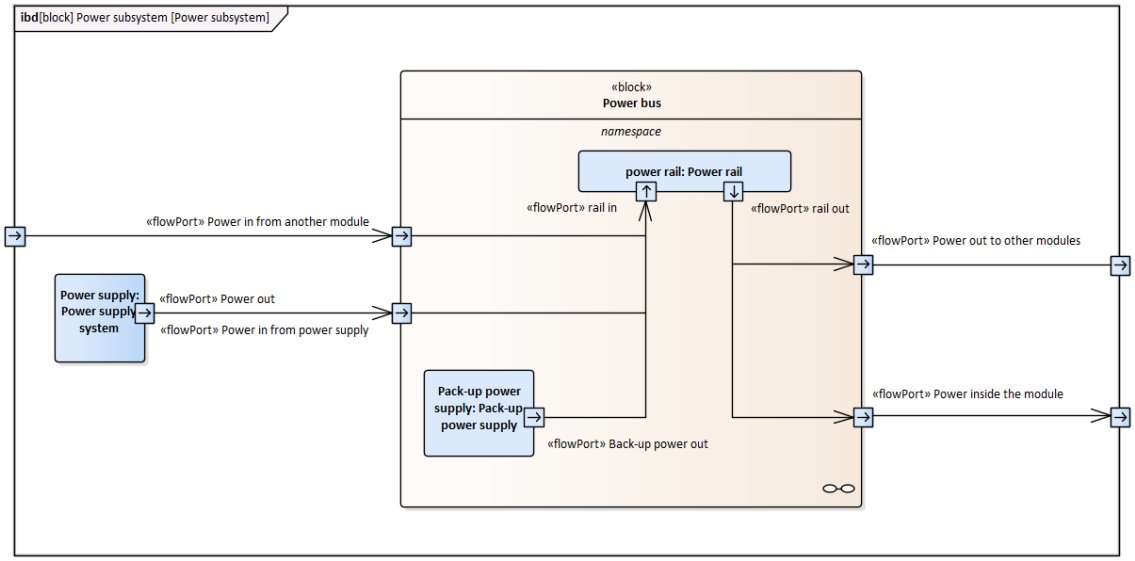


***Figure 31.*** *Sub-system architecture*

Architecture was modeled based on a design derived from the project proposal. Some parts were specified in the proposal, but most were simply guesses of what the system would look like when actual design decisions were made. This means that the diagrams shown here do not represent any concrete system and only serve to generate modeling data to be analyzed and to create a modeling framework. While the architecture should be based on the requirements and system functions, it is easy to start drafting what the system might look like by starting the modeling process from the physical architecture. This was also a problem during this case study, when there is no guideline to follow, some steps might get overlooked leading to an inconsistent model. This is what the framework attempts to alleviate by showing what should be modeled and where.

## 6.5 Traceability between model elements

At times it was helpful to go through the model piece by piece in order to find mistakes and deficits. SysML has inherent traceability between linked elements, this feature can be used to analyze the model by taking a single element and seeing what connections it has. Tools offer traceability features to a varying degree; Enterprise Architect has a simple and fast traceability feature (Figure 32) that shows element relations with textual explanation of the relation type.

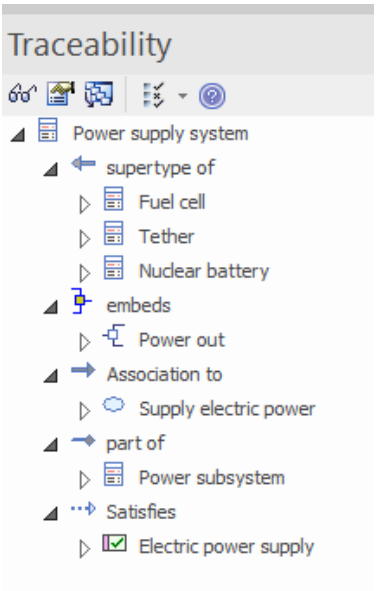


***Figure 32.*** *Traceability window*

Traceability window shows that the Power supply system is supertype of Fuel cell, Tether and nuclear battery, it has a Power out port, it is associated with “Supply electric power” use case, it is part of Power subsystem and satisfies requirement of Electric power supply. When the model is well defined traceability is an intrinsic perk.

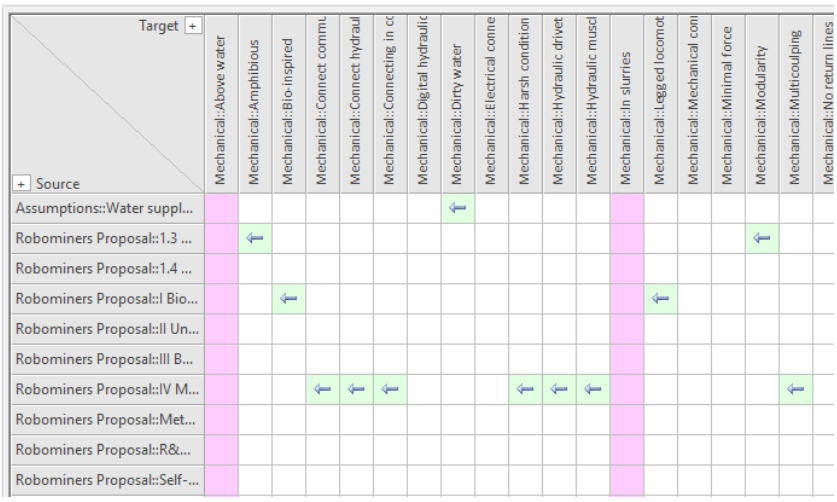


***Figure 33.*** *Relationship matrix*

Relationship matrix (Figure 33) was used to study model integrity. Whereas traceability window (Figure 32) shows all relations of one element, the matrix picks one relationship type and maps two packages with that relationship. In the example above shows mapping between requirements and requirement sources with trace relationship. Elements missing a relationship are automatically flagged, here requirements missing a source are flagged purple, directions can also be reversed by flagging sources that do not have requirements extracted from them. Other use cases for relationship matrix could be mapping requirements to satisfying system elements or system functions allocated to subsystems.

## 6.6 Model structure and allocation to the framework

The model structure was divided into packages that depict certain aspects of the system (Figure 34). Each package has a “master diagram” showing the relevant contents of the package and providing navigation links. These packages also form the core of the framework.

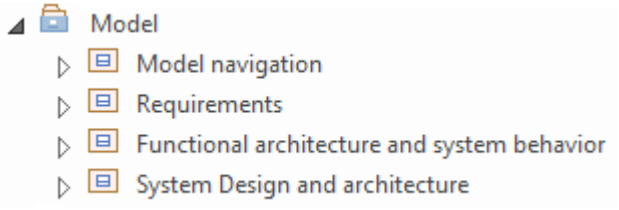


***Figure 34.*** *Model structure*

**Model navigation**

Navigation package was created to accommodate diagrams that form the framework and provide links for model navigation. The diagrams collect views from the model showing their relation to the framework and allows easy navigation between them.

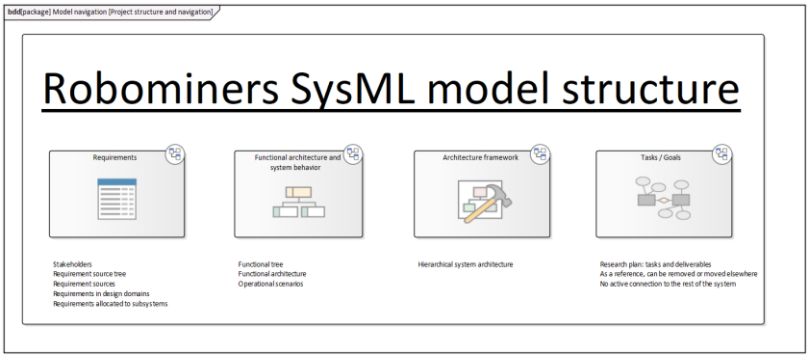


***Figure 35.*** *General navigation links*

**Requirements**

All requirement related elements and diagrams are located in the requirement package. The package contains requirement source materials, stakeholder descriptions, requirement tree and requirement descriptions. The requirements package has a diagram of the same name, that provides navigation links inside the package.

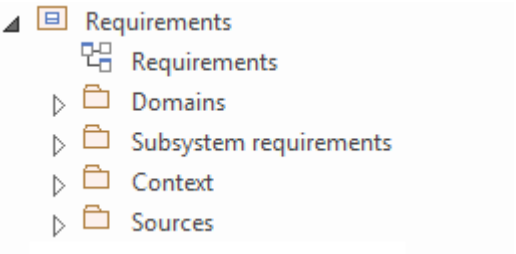


***Figure 36.*** *Model structure: Requirements*

**Functional architecture and behavior**

This package contains functional architecture and behavior descriptions for the system and subsystems. Functionality is divided into three parts: Function tree, Functional architecture, and Operational scenarios.

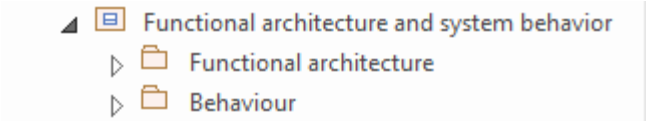


***Figure 37.*** *Model structure: Functionality*

**System design and architecture**

The actual system architecture is located in its own package. The package contains architecture descriptions of the system and subsystem on different abstraction levels.

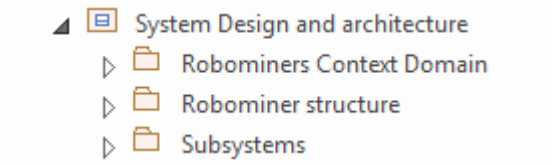


***Figure 38.*** *Model structure: Architecture*

As the modeling steps were done without a framework, the process was iterative and not well defined. The purpose of allocating the different parts into the framework is to form a well-defined guide that when followed would provide a more streamlined modeling process. The way the framework was formed was to base it on abstraction levels in order to guide modeling focus on the relevant features at a given process stage. The modeled aspects are present on all levels and grow in detail as abstraction level goes lower. The levels and modeling aspects form a grid shown on Figure 39.

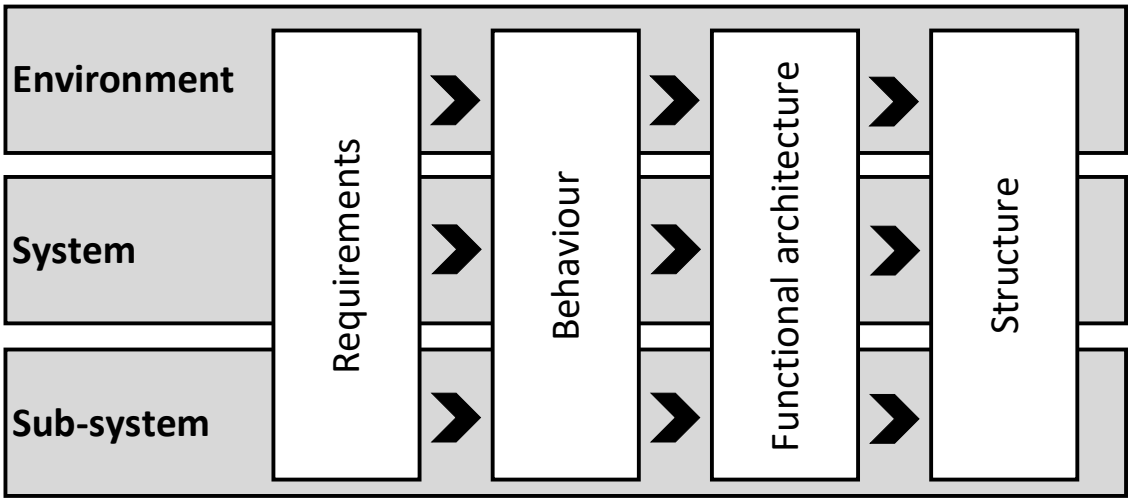


***Figure 39.*** *Preliminary framework idea*

Additional package was created for behavior models as it was separated from functions making behavior the fourth aspect in the framework.

# 7. SYSTEM ARCHITECTURE FRAMEWORK

This section presents the results of this work: system architecture framework suited for modeling robotic systems. The purpose of the framework is to guide modeling efforts and to show what diagrams and features to use, when how and why. Guidelines are needed for SysML modeling in order to keep the model well organized and to guarantee a proper coverage.

## 7.1 Scope of the framework

The framework has a hierarchical structure corresponding to different abstraction levels. Division to abstraction levels is commonly used in many frameworks ( [13], [23], [10]) and allows the system of interest to be looked at smaller, easy to comprehend, segments. Each abstraction level has a set of views for the system elements on that level. The views depict different aspects of the system. The framework views are combined and adapted from the works of Lamm and Weilkiens [15], J. Holt et al. [17] and G. Barbieri et al [13], the structure takes inspiration from BMSE Grid [10] and Consens [22].

The framework considers four main aspects: Requirements, Functionality, Behaviour and Structure. Eack aspect is viewed from internal and extenral perspective, this creates 8 view types in total. Views are listed in Table 1 and will be introduced in section 7.3. The views are applied to each abstraction level and have slight level specific nyances. The role of external perspective is to provice a hierarchical “checklist” to aid with design coverage and to create easily navigateable structure. Internal perspective shows how the elements defined in the external perspective manifest themselves in the system. In short: external identifies the elements in a given context and internal specifies the relationships or those elements.

***Table 1.*** *Framework views*

| | **External (Identify)** | **Internal (Specify)** |
|---|---|---|
| **Requirements** | Requirement context view | Requirements view |
| **Functions** | Function tree view | Functional architecture view |
| **Behavior** | Scenario list view | Scenario view |
| **Structure** | Breakdown view | Architecture view |

Table 1 lists the general views to be used on each abstraction level. The views take slightly different roles and forms on different abstraction levels, but their purpose remains the same.

## 7.2 Abstraction levels

To help with the modeling of the system the model is divided into several abstraction levels (Figure 40), the structure follows that of a V-model. MBSE process can be generalized to a V-model, when the framework follows a familiar structure, its deployment on a project becomes less demanding. Each abstraction level contains multiple views that show different aspects of the system related to that level. It should be noted that each level might have multiple elements that each have individual views. For example, in subsystem level each subsystem must be considered individually. And if the system is composed of two systems, both systems are to be considered on the system architecture level.

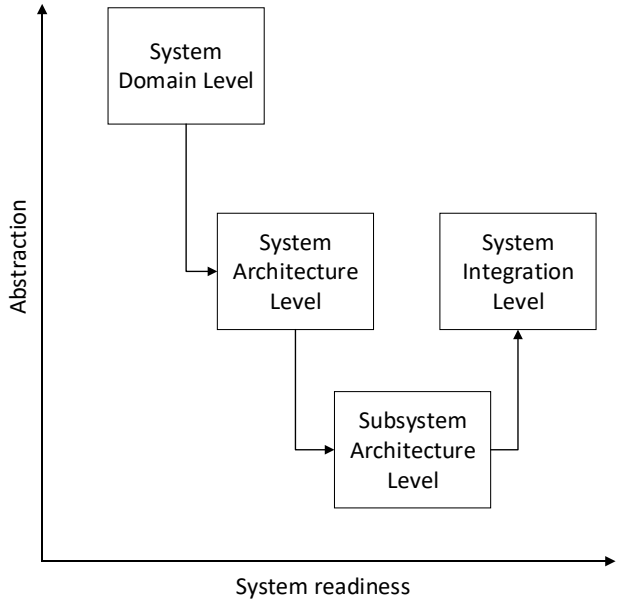


***Figure 40.*** *Hierarchical framework*

When looking at the system on a certain abstraction level, all relevant elements are considered in same level of detail. Level of detail in relationships and interfaces between elements is also dependent on the abstraction level, on higher abstraction the relationships are general, and elements have a black box representation. As abstraction goes lower and system readiness increases, the relationships get more specified and elements move to a white box representation. The framework starts from general ideas and problems and divides them into smaller concepts. This means that as the abstraction level decreases, the number of system elements increases. One benefit of this structure is to make the diagrams easily accessible. When all diagrams related to a certain hierarchical level are collected to one place, they are easy to find when needed. The diagrams are collected into block definition diagrams using hyperlinks or navigation elements if supported by the selected SysML tool. Each level acts as a black box presentation of the lower level and as a white box presentation of the upper level (Figure 41).

The abstraction levels shown on Figure 41 imply a modeling order, but the process can also be iterative or parallel depending on the project needs.

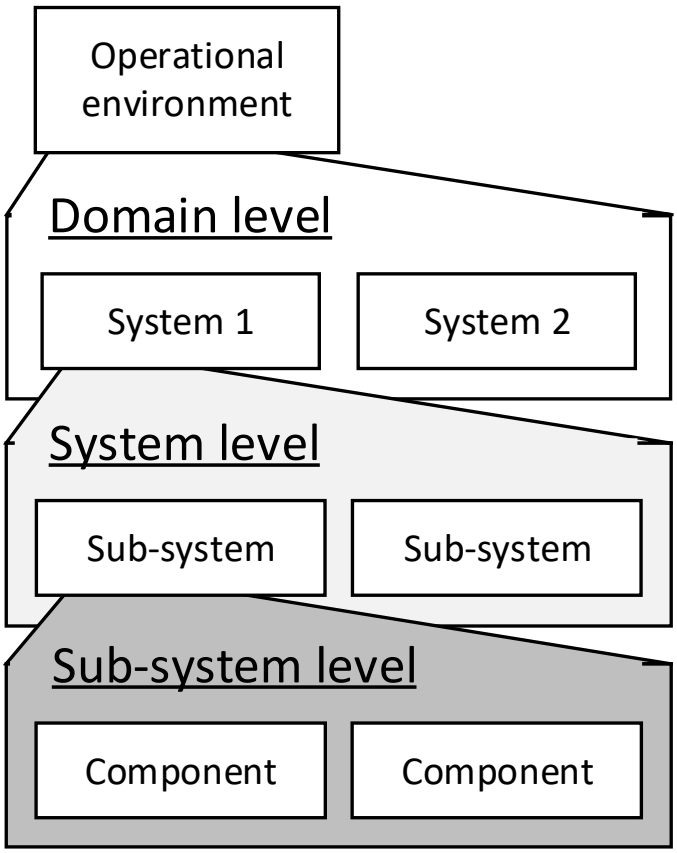


***Figure 41.*** *Level relationships*

**System domain level** starts from defining the environment the system operates in and identifying the interfaces with other systems in the environment. High level functionality is also considered in the domain level.

**System architecture level** continues from the domain level by decreasing the abstraction level and giving more specific views of the system itself, its subsystems and functionality.

**Subsystem level** considers each subsystem defined in the system architecture level and specifies the parts and functionality of the subsystems.

**Integration level** increases the abstraction level by combining the subsystems into one system and formalizes the interfaces between subsystems. As Integration level is on the same abstraction level as the system level, it is omitted from Figure 41.

### 7.2.1 System Domain level

No system exists in a vacuum, the environment influences the system and its design. There are interactions between other systems and environmental elements that affect the design and should be taken into consideration. The system domain level addresses this by modeling the system of interest as a part of its operational environment. The word “environment” is used in its broader meaning and includes all possible physical environments, laws, regulations, and other actors that are present in a specific operational space, the overarching domain. As the highest abstraction level, the domain level considers the system as a singular entity that has all the required functionalities to satisfy the stakeholder needs and comply with laws and regulations. Domain level might not be useful for documenting the system itself, other than to establish interfaces with other systems, but it offers a starting point for the design. The high-level functions and requirements identified on this level are then resolved on lower abstraction levels.

Requirements on this level are considered by identifying the stakeholders and their needs regarding the system and its functionalities. Requirements given by the stakeholders are usually not technical or formal, hence they cannot be considered ready nor validated system requirements. Stakeholder requirements are used as source material in specifying the system requirements.

**Requirements**
- External: Specify stakeholder “stakes” on the system
- Internal: Capture requirement source material

**Behavior**
- External: Identify key operational scenarios
- Internal: Model the behavior flow / sequence

**Functions**
- External: Identify main purpose of the system and key functions
- Internal: Rough draft of functional dependencies

**Structure**
- External: Identify the actors on the specific domain
- Internal: Model interactions between the actors and environment

### 7.2.2 System Architecture level

System architecture level is one step lower in abstraction than domain level. This level shows the composition of the system of interest and the relationships between its subsystems. Purpose of this level is to identify all the subsystems and connections between them that must be considered in further detail on lower abstraction levels. Subsystems have to communicate with each other, these connections are addressed on this level, defining formal interfaces might not be feasible nor necessary on this level, but their presence is still addressed and documented in some detail.

System architecture level is a high-level presentation of the system and answers to the challenges brought up in the domain level. The high-level functions and stakeholder requirements from domain level are considered in greater detail. System level functions are divided into smaller functions and allocated to subsystems similarly stakeholder requirements are turned into system requirements and then allocated to subsystems.

This level aims to establish the groundwork for the design process, creating a reference / blueprint that can be used to guide the design efforts. The requirements and specifications are set on this level for the upcoming design phases.

**Requirements**
- External: Create a requirement classification
- Internal: Specify system requirements derived from the source material

**Behavior**
- External: Extend the scenarios identified on domain level
- Internal: Model the extended behavior flow / sequence

**Functions**
- External: Identify system level functions
- Internal: Model functional dependencies and architecture

**Structure**
- External: Identify subsystems
- Internal: Model the system architecture

### 7.2.3 Subsystem Architecture level

Subsystems listed on the system architecture level are considered individually on the subsystem architecture level. Each subsystem is given a set of views that define their composition and interfaces. This is the most extensive part of the model as there are multiple subsystems that have to be considered individually and in detail. This level works the same way as other levels viewing the subsystem as the focus for the level. Some subsystems might have to be divided further into deeper levels; these levels can be added on the structure in the same way as other levels.

Sub-systems are presented as black-boxes on system architecture level, the sub-system level considers the requirements, interfaces and functionality defined in the system architecture level and creates a white-box presentation for each subsystem. As part can have multiple ports with different properties it is important to document those ports and their connections. On upper levels, these connections can be handled by ignoring the ports as the connections might not be fully specified.

**Requirements**

| | |
|---|---|
| External: | Give context to the requirements by linking to functions / use cases |
| Internal: | Allocate system requirements to subsystems |

**Behavior**

| | |
|---|---|
| External: | Identify / allocate scenarios to the subsystem |
| Internal: | Model behavior flow / sequence |

**Functions**

| | |
|---|---|
| External: | Identify sub-system specific functions |
| Internal: | Create subsystem functional architecture |

**Structure**

| | |
|---|---|
| External: | Identify subsystem components |
| Internal: | model sub-system architecture |

### 7.2.4 Integration level

This level can be seen as a first upwards step on a V-model and is on the same abstraction level as system architecture level. Integration level defines more formal interface connections between the different subsystems. on system architecture level the architecture can only be defined in a low detail as the interfaces are not well known on that stage of development. More detailed interfaces are created later, on sub-system level and these interfaces are then “updated” to the system architecture level. These updated views are created on separate diagrams and are collected on the integration level that concentrates on architecture views, connecting the subsystems on all domains (mechanical, electrical, software) with more formal notation.

As integration level and system architecture level are on the same abstraction level, they should be essentially the same. Whereas the system architecture level set specifications and requirements for the design, the Integration level is result of the development done since then and should reflect the design decisions done in that time. Hence this level can be used to check if the design has stayed within the boundaries set on the system architecture level and how well the current design accomplishes to realize the architecture.

**Requirements**
Does the integrated structure meet system requirements?
**Behavior**
Has the planned system behavior changed?
**Functions**
Does integrated structure produce wanted functions?
**Structure**
Are the subsystems compatible?

## 7.3 Framework views

This section goes over the different views that can be used. Using all views for every part might not be necessary, breakdown-, architecture-, and requirement views are the most useful. Views are based on different SysML diagrams.

### 7.3.1 Requirement Context View

Requirements: External

**Concern**: Give context to requirements

This view is borrowed from ACRE-framework authored by Jon Holt et. al. [17]. They show that there might be moments where multiple stakeholders have the same requirement, but it affects the system differently based on context. By looking at the requirement from different points of views, additional insight and understanding might be attained. For example, requirements “reduce costs” is vastly different from the supplier and customer perspective. Requirements Context View refines requirements by placing them in the context where they manifest themselves. Use cases are connected to requirements with SysML refine relationship. This view is useful when there is ambiguity in what point of view the requirements should be looked at or where they affect the system. Context view is created with a use case diagram. Depending on the abstraction level the view serves a slightly different purpose.

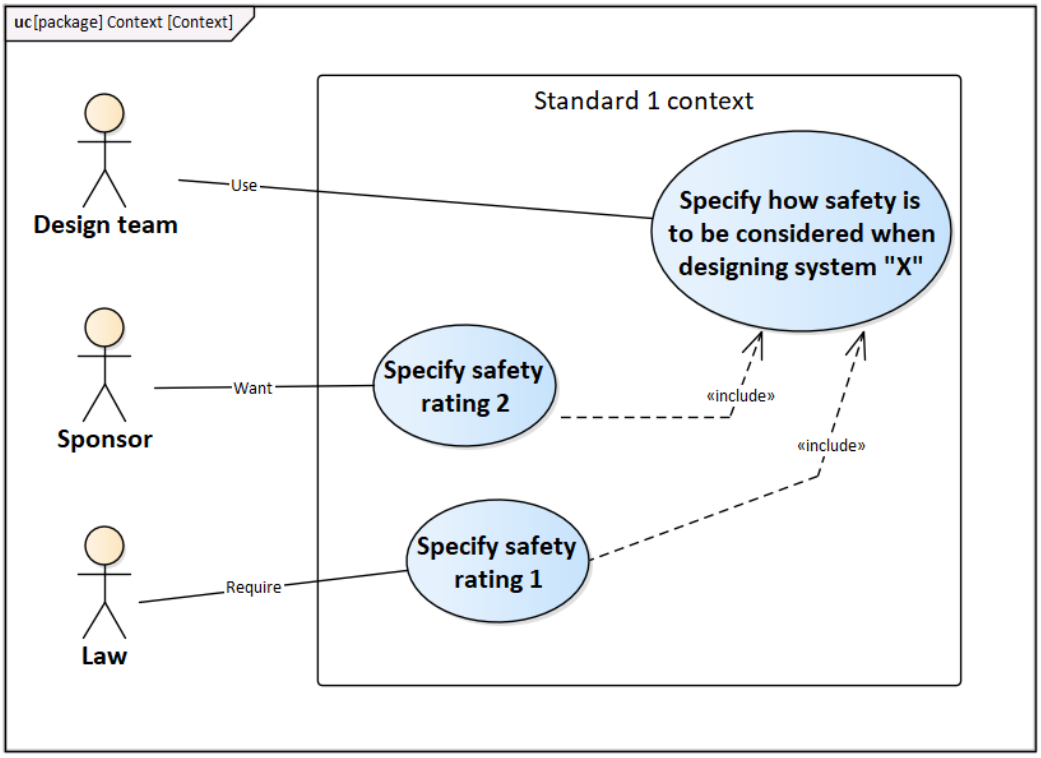


***Figure 42.*** *Stakeholder context view*

On domain level the view is used to show what interests different stakeholders have in the systems and their relation and significance is to the project. Figure 42 is a simple example of this view showing a stakeholder context of an unspecified standard and its relation to the other stakeholders. In the background the use-cases are linked to requirement sources with the purpose of giving a short description of their content and intent. Each stakeholder is linked to a folder containing their respective requirement source material. On system level the view shows how the system answers to the primary stakeholder concerns and needs. Similarly, on sub-system level the view is used to show what roles different sub-systems have in satisfying system requirements. On all levels the purpose of the view is to provide a rough outlook on the purpose of a specific actor of system part and show how and where the requirements manifest themselves.

### 7.3.2 Requirements View

Requirements: Internal

**Concern**: Identify system requirements

Requirement view (Figure 43) shows the requirements of the related element. The view is a hierarchical tree structure that starts from the requirement source element that the requirements are elicited from. The requirements are connected to the source with a “trace” relationship, a dashed line arrow. The first requirement is the primary higher-level requirement and the sub requirements are linked with a “containment” connector, circle with a cross and a line. If a there is a system element that satisfies the requirement it can be connected with a “satisfy” relationship. The satisfying element is not created for the diagram but linked from a diagram that defines the element, (breakdown view).

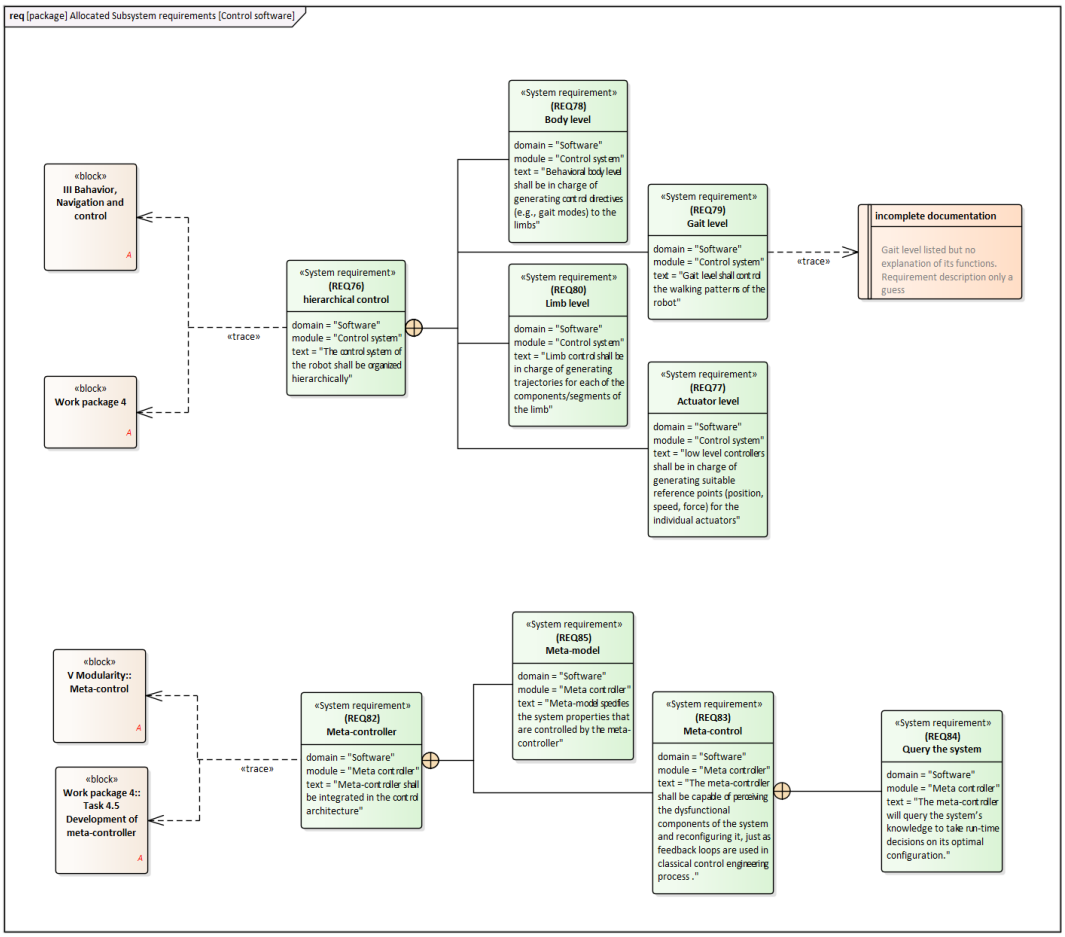


***Figure 43.*** *Requirement view*

To help with the readability of the diagram, some relations are hidden. The underlying model and the diagrams are separate entities that are linked to each other, diagrams are not the model itself. For example, all requirements should be linked to the source, but the links do not have to be in the diagram. In Figure 43 only the primary requirement has a drawn connection to the source element, others are hidden, but can be assumed to be from the same source as the primary requirement.

### 7.3.3 Scenario list View

Behavior: External

**Concern**: Identify operational scenarios

The purpose of this view is to lists different operational scenarios to be modeled and offer links to the modeled scenarios. Each scenario links to an internal Behavior View diagram that shows the actual activity flow. On domain and system level the scenarios modeled are derived from the requirements and the projects mission statement. The goal of modeling behavior is to identify what steps or actions the system must take in order to accomplish its task. System's end goal is specified in the requirements, but the way to get there is usually not as there can be many ways.

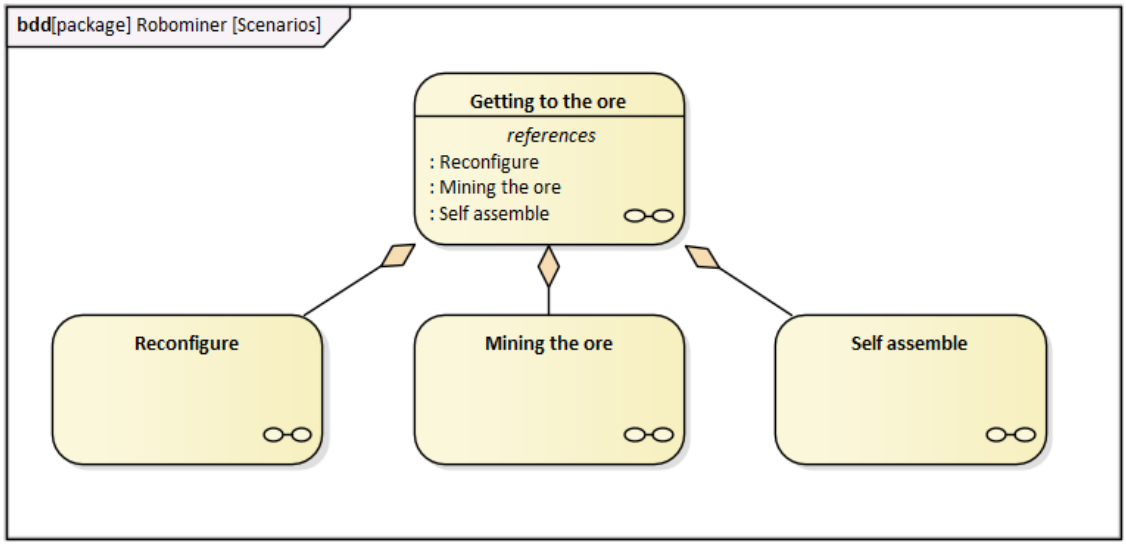


***Figure 44.*** *Scenario list view*

The listed scenarios can have sub-scenarios that can also be added while modeling the upper scenario if an included part is deemed significant enough to be modeled independently.

### 7.3.4 Behavior View

Behavior: Internal

**Concern**: Identify what steps must be completed to accomplish a task.

Behavior View continues from Scenario List View by modeling the actual scenarios it had identified. The view (Figure 45) shows the element behavior or functionality in form of an activity diagram. This view is used as a tool for designing, documenting, and analyzing the system functionality and can help with communicating with the software design team when designing the system behavior patterns.

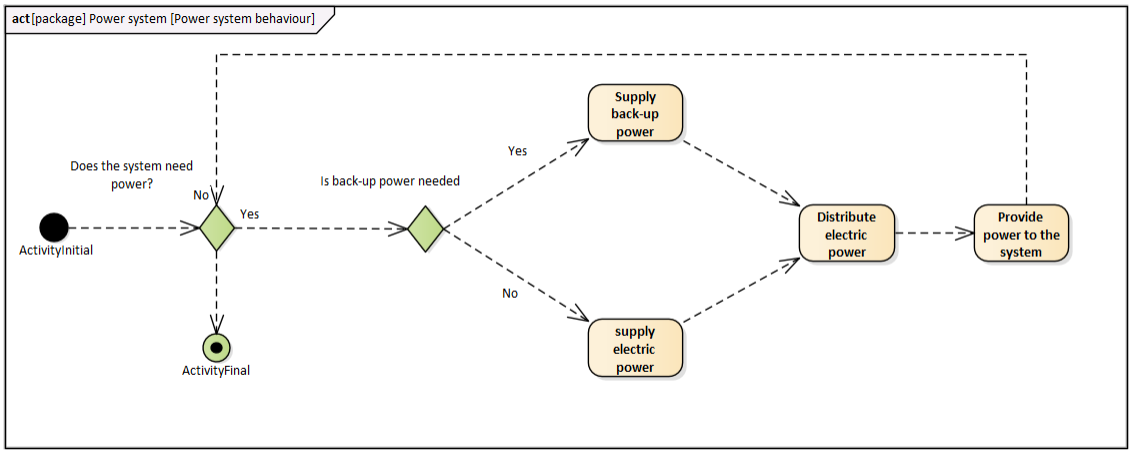


***Figure 45.*** *Behavior view*

Activity and sequence diagrams are both behavioral diagrams that have overlapping purposes. Both can be used to represent passage of events. Sequence diagram is more formal and is used to show interaction and passage of time between multiple system actors. Activity diagram can serve the same purpose with a swim lane presentation.

### 7.3.5 Function Tree View

Functions: External

**Concern**: Identify system functions

Functional architecture (Figure 46) is used to define structure for the robot functionality. Function tree acts as a breakdown view for functionality, showing what functions should be addressed in the design. Function tree is modeled as a block definition diagram using activity blocks.

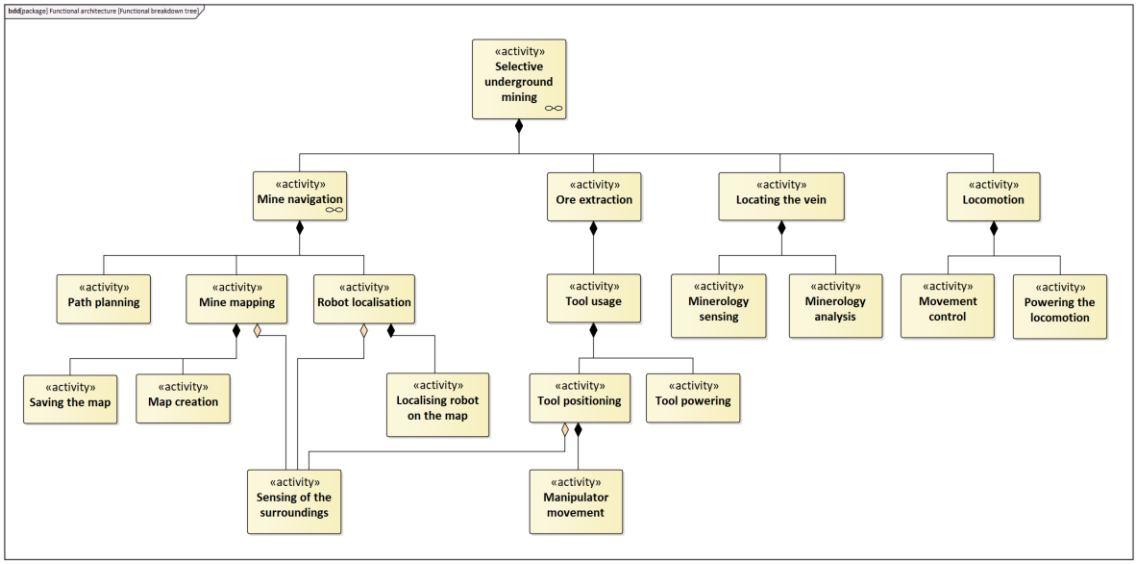


***Figure 46.*** *Function tree*

Function tree is created based on functional requirements. Use cases that refine functional requirements (Figure 42) can be a helpful tool to derive functions form, this is an approach recommended by Lamm and Weilkiens [15]. They also note that functional tree only defines hierarchy not necessarily composition, this means that function is always executed in the context of its parent function but doesn't always belong to the same logical group as other functions with the same parent.

### 7.3.6 Functional Architecture View

Functions: Internal

**Concern**: Specify functional dependencies.

Functional architecture shows relationships and structure between system functionalities. Functional architecture should not imply physical implementation of the system, its structure should have no ties to physical world. This means that a functional architecture can have multiple alternative physical implementations; one functional architecture can be used across generations and different technological limitations [15].

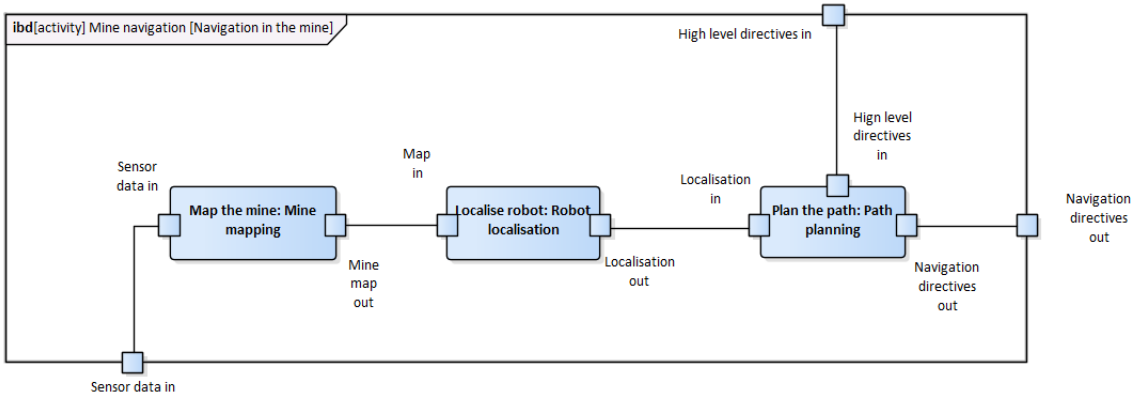


***Figure 47.*** *Functional architecture of a navigation software*

Functional architecture forms a network of connections and functions that can be grouped. Functional group is a collection of functions that are closely related with inputs and outputs. Groups should be as independent as possible; this can be used as a grouping criterion by forming groups so that they have minimal interaction with other groups. Lamm & Weilkiens [15] note that functions cannot be grouped systematically because functional architecture combines technology and art and requires thought.

Figure 47 Shows an example of grouped functions to form architecture of a navigation software. The group has three interaction points that link it to other groups, these groups form the functional architecture for the system. The complete functional architecture can be visualized by using the groups as individual blocks.

The form of the functional architecture does not have a direct connection to physical architecture, functions can be realized by multiple system elements simultaneously or one subsystem can have multiple functions. Functional architecture is realized as a physical architecture that has functions allocated to specific subsystems.

### 7.3.7 Breakdown View

Structure: External

**Concern**: Identify system elements on a specific abstraction level.

Breakdown view (Figure 48) shows the composition of a system or an element in a form of a block definition diagram. This view can be seen as a hierarchical list of parts. Breakdown view does not consider how the listed elements relate to each-other, it simply tells what parts make the system. Breakdown view is used for every system and subsystem in the architecture in order to define their composition.

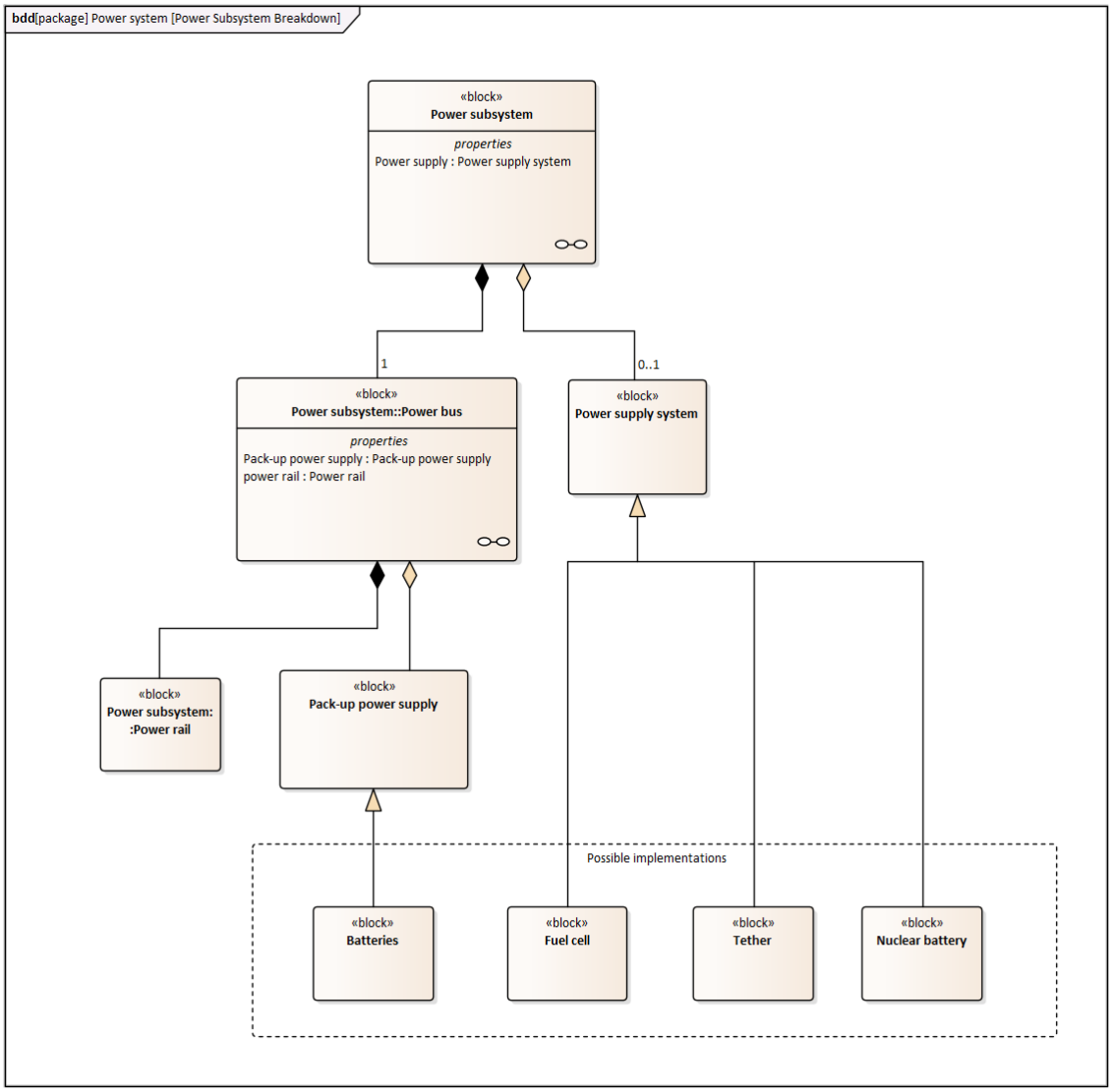


***Figure 48.*** *Breakdown view*

Breakdown views are used to define elements that are used in architecture views. As elements are defined on breakdown view before being used in architecture view, the view can be used as a checklist to see if all elements are being considered. Hierarchical structure also provides a clear and quick way to navigate the model.

### 7.3.8 Architecture View

Structure: internal

**Concern**: Specify relationships between system elements on a specific abstraction level.

Architecture view is related to the breakdown view and shows how the parts listed in the breakdown view relate to each-other, this is done in form of an internal block diagram. Depending on the use, the view can be just a general logical architecture, or it can specify formal interfaces between the subsystems and parts.

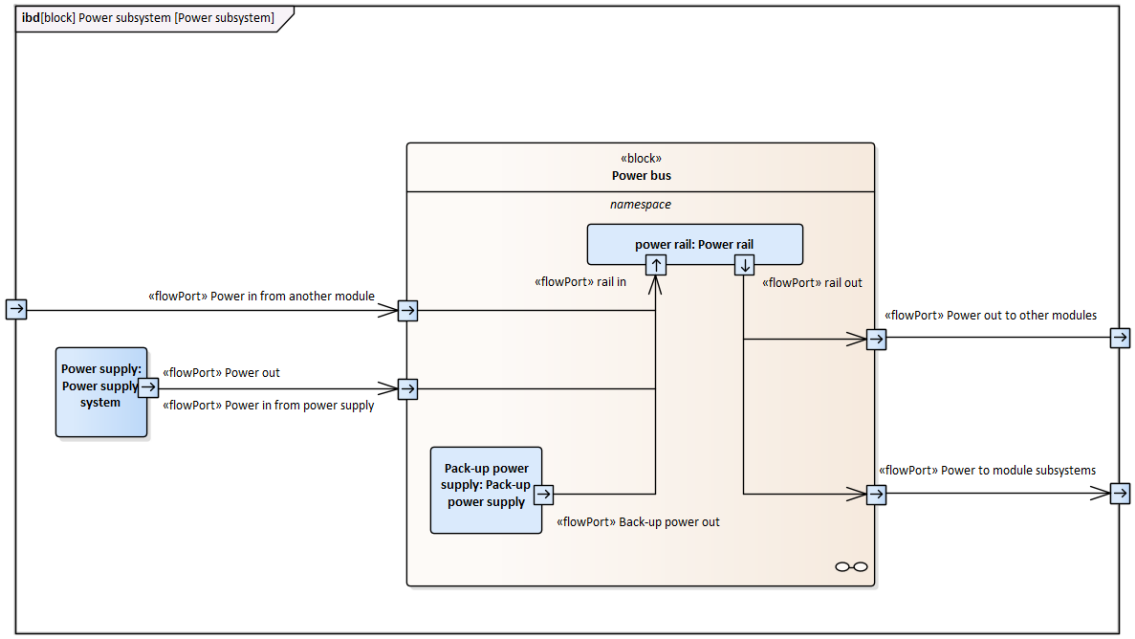


***Figure 49.*** *Architecture view*

Ports and written relations can be used to define interfaces. Example above (Figure 49) shows a logical architecture without hardware interfaces. Multiple views can be created to capture different types of interface domains. The level of detail depends on the purpose of this view. The purpose changes depending on the domain type (mechanical, electronic, software…) and the hierarchical level and readiness of the design.

### 7.3.9 Additional views

In the background several navigation views can be created such as categorization for stakeholders providing links to their context and requirement views (Figure 50).

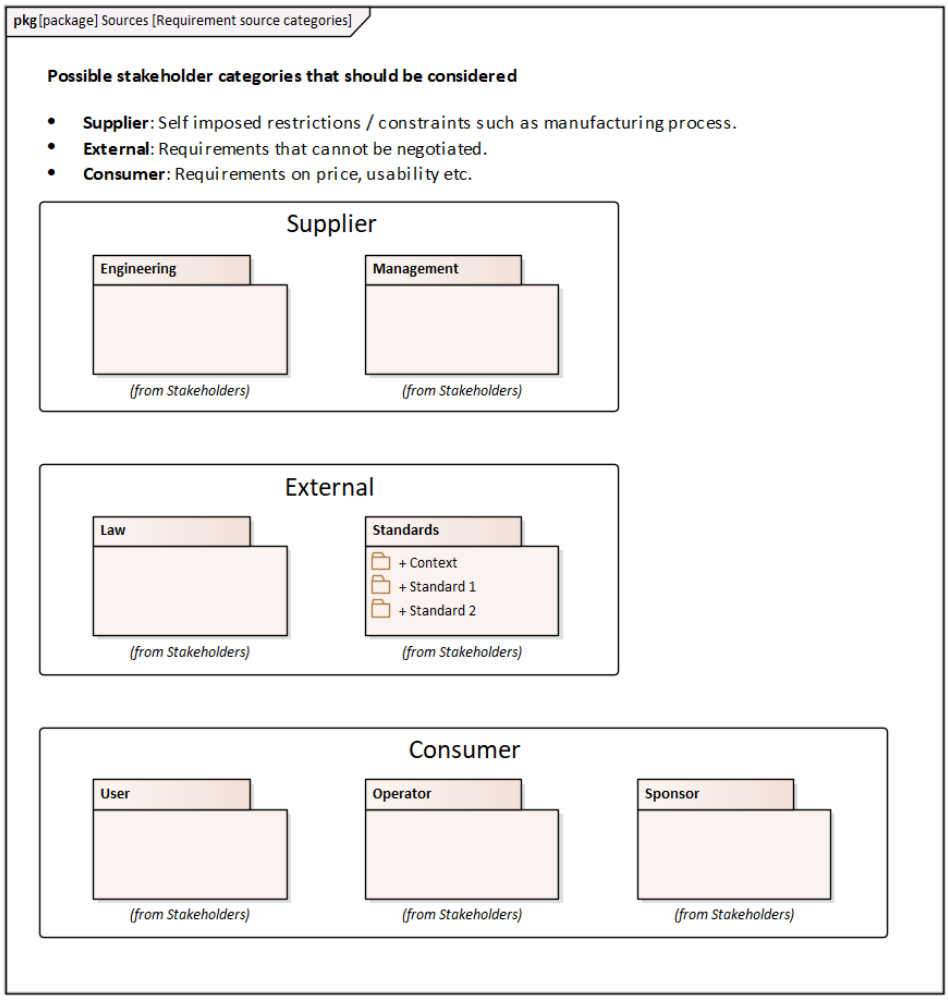


***Figure 50.*** *Source categories*

These views are not part of the framework itself but offer useful links for navigation purposes. It is also beneficial to create a view showing a bigger picture of the modeling process, acting as a checklist. Figure 51 shows an example of System Level, when something on the framework is modeled links to it can be added to a navigation view.

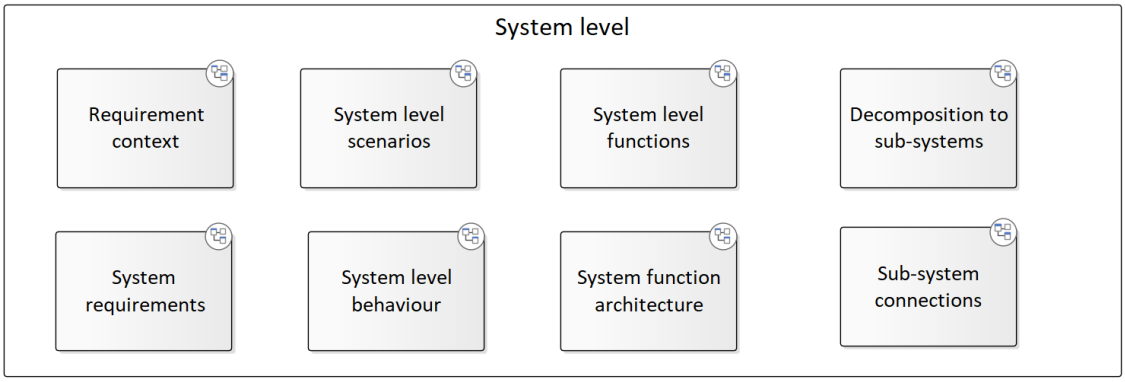


***Figure 51.*** *Navigation example*

On the domain level the requirements view is replaced by a requirement source view (Figure 52). The purpose of this view is to include the raw requirement source material in the model so that it can be traced to the requirements that are derived from them. For example, the source material can be user stories, stakeholder requirements, standards, and laws. Depending on the modeling tool used there can be several ways the source material can be included into a model.

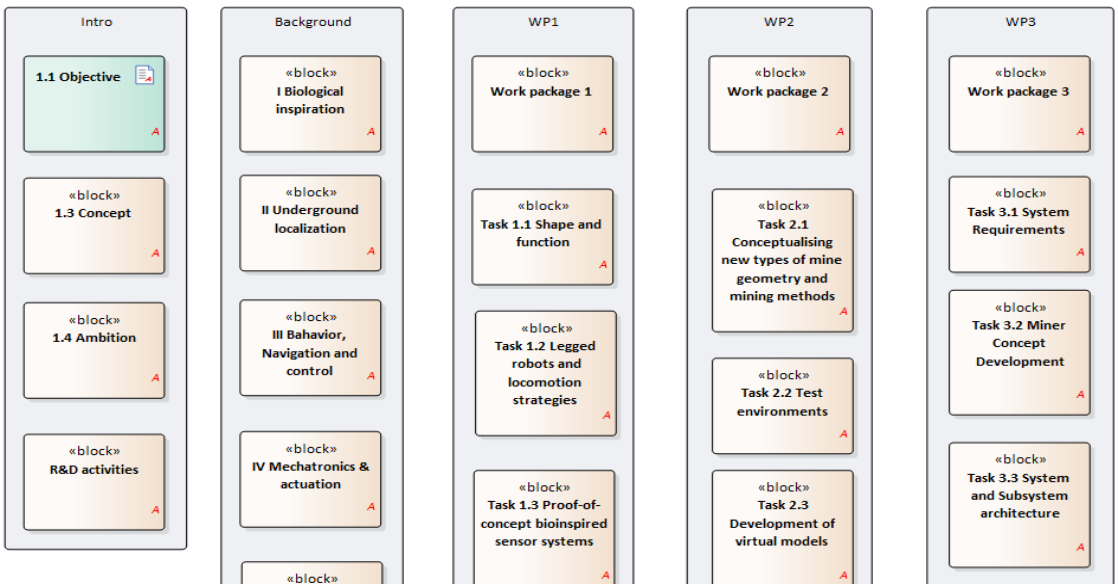


***Figure 52.*** *Requirement source view*

In Figure 52 the source elements are document blocks that enclose a text document containing the source material. Another way to include sources is to use hyperlinks pointing to the original text- or pdf-file.

## 7.4 Metamodel and process example

The metamodel in Figure 53 shows the possible relationships between element types and gives an outlook on how the framework structures a model. The metamodel also lays out a modeling order. Relation arrows point toward a higher abstraction. Highest is the stakeholder classification that categorizes different stakeholder. This is done with a Source Tree diagram shown in Figure 19 that puts different stakeholder into categories based on their role on the project. Each stakeholder has a folder for stakeholder requirements that are documented as requirement sources. System requirements are extracted from the source documents and visualized as a requirements diagram. Behavior is modeled based on functional requirements in order to derive the needed system functions and to close possible gaps in the requirements. Functional architecture is constructed from the system functions and a system architecture is designed based on the functions. Use-case diagrams are used to visualize how system elements satisfy the requirements.

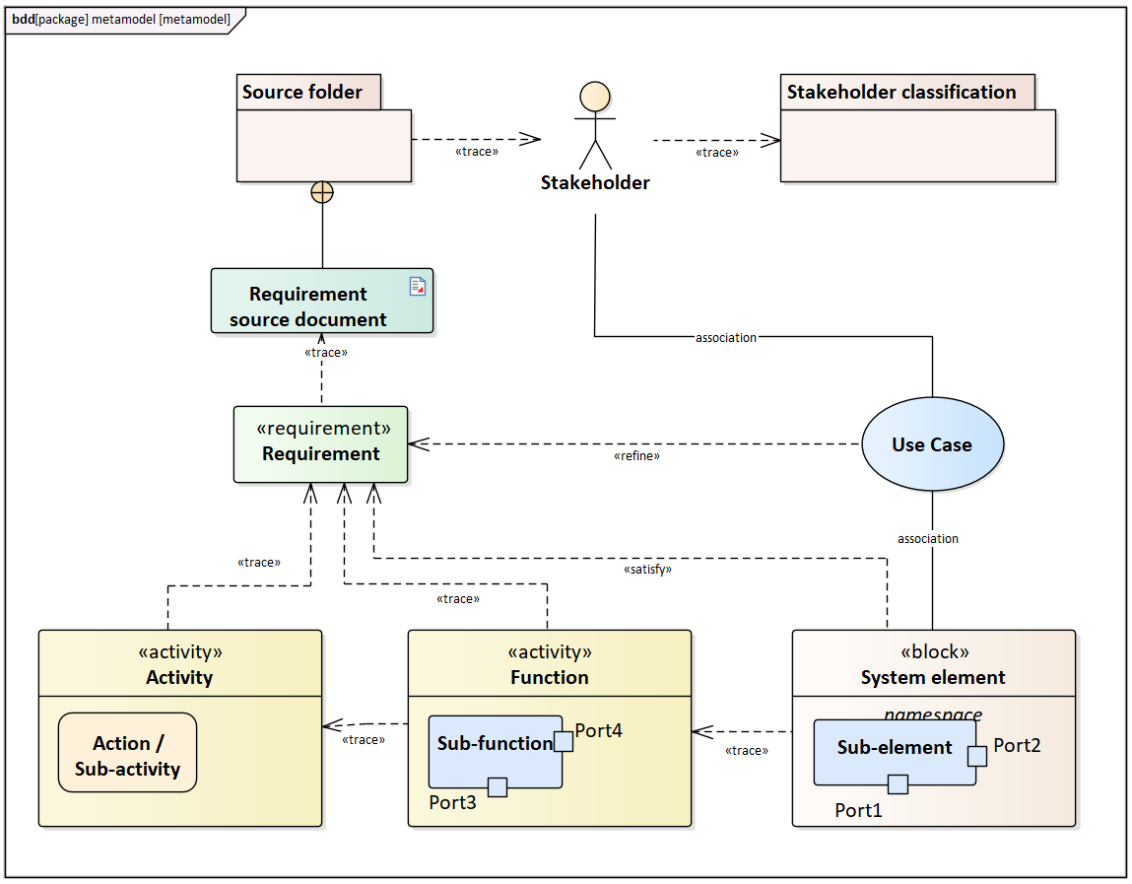


***Figure 53.*** *Framework metamodel*

There are also relations between elements of a same type, but these have not been visualized in the metamodel but can be seen in their respective view descriptions.

### 7.4.1 Example modeling process

The process assumes that mission statement and stakeholders are known.

`[R-E]` = Requirements External
`[R-I]` = Requirements Internal
`[B-E]` = Behavior External
`[B-I]` = Behavior Internal
`[F-E]` = Functions External
`[F-I]` = Functions Internal
`[S-E]` = Structure External
`[S-I]` = Structure Internal

**Domain level**

1. `[R-I]` Collect stakeholder requirements
2. `[R-E]` Create a context view for each stakeholder
3. `[S-E]` Define the operational environment of the system
4. `[S-I]` Model system's interactions with environment and external systems
5. `[B-E]` Identify main operational scenarios (system as an "actor")
6. `[B-I]` Model the identified scenarios
7. `[F-E]` Identify the primary functions
8. `[F-I]` Model relations between the functions

**System level**

1. `[R-I]` Derive system requirements from the stakeholder requirements
2. `[F-E]` Identify system level functions
3. `[F-I]` Model the dependencies and interactions between the functions
4. `[S-E]` Identify the needed sub-systems to realize the system functions
5. `[B-E]` Identify system level behavior scenarios (sub-systems as "actors")
6. `[B-I]` Model the system level behavior
7. `[S-I]` Model interactions between sub-systems
8. `[R-E]` Create a system requirements context

**Sub-system level**

1. `[R-I]` Allocate system requirements to sub-systems
2. `[R-E]` For each sub-system create a requirement context
3. `[F-E]` Allocate system functions to sub-systems
4. `[B-E]` Identify sub-system level behavior scenarios (components as "actors")
5. `[B-I]` Model sub-system behavior
6. `[F-I]` Extend the functions and model sub-system function architecture
7. `[S-E]` Identify the sub-system components to realize functional architecture
8. `[S-I]` Model interactions between components

# 8. FUTURE WORK AND DISCUSSION

This work set out to create an architecture framework that provides an organized modeling structure and captures all aspects of a mechatronic system. The goal was to be able to provide a framework that is easily adaptable to a design process and gives a guideline for SysML modeling. The framework presented in this work is derived from literature and adapted during a case-study. It has not been tested on an actual application, so its practicality, adaptability and scalability remain uncertain. Every project is different in some way, some features of the framework might not be needed and might simply create confusion. Some of the least useful features could be removed or revised after testing and feedback, however that was not in the scope of this work. The framework is designed to work on a mechatronics design process, and should support common MBSE methodologies, but as the framework has not been tested, this claim cannot be confirmed. Another case study should be done to verify that the framework serves its purpose and works as intended without any major issues. If applied on an actual project the framework will most likely have to be refined during deployment. At its current state It should still provide a good starting point for implementing a SysML based modeling methodology.

SysML models themselves can be useful to every project, but creating the model takes effort and time. In some projects that time could be utilized on something more useful. Familiarizing the design team to SysML and allocating work hours to training and upkeep of the model might not be feasible on some smaller scale projects. In bigger projects, SysML or other model-based tools can become essential to guarantee success. This in mind, it is important to clarify what the role of modeling is in the project and what modeling techniques to use, meaning that a modeling guideline should be implemented. While a model created without guidelines might not be inherently incorrect from a technical point of view, it might have parts with differing structure created by teams using different methodologies [21]. This might not cause issues on a smaller scale research project as the process does not have to be replicated but means that the model cannot be easily reused. Model and process reusability can become an important aspect for bigger organizations with multiple projects similar in scope and domain. In a small homogenous group that works together, SysML might not provide tangible benefits as everyone is familiar with the system and know each other’s contribution to the design process. When complexity increases and more people on multiple sites with different backgrounds are involved in design, more formal documentation methods are needed, in these situations SysML starts to show its merits.

While the framework itself is not dependent on a certain modeling tool or even on SysML it was created using Enterprise Architect (EA) and might contain features that are not easily replicable with all existing modeling tools. The framework was created using mostly standard SysML features but some adaptation might have to be done when recreating the framework with another tool. The framework uses hyperlinks and internal navigation links that might not be supported by all available modeling tools. There are also some issues that are present even with EA. When creating a diagram, EA provides a default toolbox for the associated diagram type, however in some cases the features used in the framework are not in the default toolbox. This is mainly a problem when creating links between elements in multiple diagrams of different type. This can cause confusion and requires the user to find the feature from a larger selection, this can be averted by creating a custom toolbox that includes all relevant features.

The framework does not have formally defined verification rules that could be used to check for the model integrity. These rules are somewhat dependent on the specific project and process the framework is used at, but the starting point is that links between elements should be maintained all the way from requirements to the final design. Metamodel in Figure 53 shows the expected relations between element types, note that the arrows point towards a higher abstraction level and ultimately lead to requirement sources. The metamodel in a way implies verification rules and can be used as a reference when building a model.

# 9. CONCLUSIONS

The framework presented in this work was created to support design efforts of mechatronic robotic systems and to provide a guideline for SysML modeling. The framework provides a starting point for SysML based MBSE process reducing time investment and making adoption easier. The following issues were the driving factors of this work when formulating the framework:

- Integration: All parts and sub-systems must function together.
- Coverage: All sub-systems must be addressed.
- Evolution: Parts of the system may change during development.

To address these issues, the following questions had to be answered:

1. **What are the most useful SysML usage modes in mechatronics design?**

   SysML offers a standardized form of documenting the system architecture. SysML supports all essential aspects of documenting the system including requirements and behavior on top of architecture modeling. SysML model has inherent links that tie parts of the system together, making the architecture easy to navigate. The ease of navigation and inclusion of requirements helps with analyzing the system during and after development by allowing designers to trace back different design decisions.

2. **How SysML can be applied to different phases of a design process?**

   Following the V-model and abstraction levels on the framework, SysML can be used during the entire project life cycle from requirements to design, integration, and validation. Operational environment and interactions with other systems can be modeled giving an overview on the scope of the project. Requirements can be documented and linked to the rest of the system creating a structure that allows tracing back on the design decisions made during development, giving developers a tool to analyze and validate their design. The system architecture can be modeled and used as a blueprint to guide further development and to enable interdisciplinary communication and collaboration.

3. **What are the key aspects that should be included in an architecture model?**

   Requirements, Behavior, Functions, Structure. These were identified to be the core components needed to create a system architecture while addressing the previously mentioned issues.

**4. How an architecture model should be structured?**

The framework uses a hierarchical structure that starts from defining the operating environment of the system of interest and continues towards a higher detail level and lower abstraction by dividing the system into sub-systems and creating interfaces, links and traceability between them.

While the primary function of the architecture framework is to define a system architecture, it is not limited to the architecture itself. The framework is built to ensure design coverage by including the steps needed to create the architecture. All four aspects (Requirements, Behavior, Functions, Structure) work towards an architecture that considers all necessary sub-systems and provides the required functionality. The framework uses requirements to drive the model, all modeled parts are directly or indirectly linked to requirements and/or rationale to justify the part's existence. Functional requirements combined with behavior modeling ensure that all system functions are being considered.

The framework uses three abstraction levels (domain, system, sub-system) that show the system on different detail levels. This helps designers to focus on the relevant parts of the system at a given design phase. Domain level defines the system's operational environment, interactions with external systems and identifies stakeholders and their requirements. On system level the stakeholder requirements are refined into system requirements and system functionality is derived from functional requirements and system behavior. Subsystems are identified, their interactions modeled, and a high-level architectural model is created, showing system structure and rough interfaces between different sub-systems. Sub-system level shifts attention to individual subsystems and defines their internal structure, behavior, and functions. Finally, integration views are created by going back to system level and connecting the sub-systems with the detailed interfaces created on sub-system level.

The envisioned usage of the framework concentrates on the left side of the V, that is the system definition phase, but the resulting model has a structure that can be used for validation activities as well. The different levels and diagrams, while considered separately, are still linked to each other. SysML's inbuild traceability retains links between system elements regardless whether those links are visualized on a diagram or not. As all links are available, a traceability tool can be used to uncover the design decisions and dependencies of the elements. This is particularly beneficial when doing changes to the design and enables verification and validation activities.

Although the original intent was to create a methodology independent framework for documenting system architecture, the resulting product heavily implies a certain modeling process to be used. However, the framework is structured similarly to a V-model, so it fits a generic MBSE process.

# REFERENCES


[1] M. Hause, "How to fail at MBSE," International Council on Systems Engineering, 2013.

[2] A. Albers and C. Zingel, "Challenges of Model-Based Systems Engineering: A Study towards Unified Term Understanding and the State of Usage of SysML," in *Smart Product Engineering*, 2013, pp. 83-92.

[3] L. Wang, M. Izygon, S. Okron, L. Garner and H. Wagner, "Effort to Accelerate MBSE Adoption and Usage at JSC," in *Space 2016 AIAA*, Long Beach, CA; United States, 2016.

[4] J. O. Grady, "Univerlas Architecture Description Framework," in *Systems Engineering Vol. 12, No. 2*, 2009, pp. 91-116.

[5] K. Hambelton, D. Kirkpatrick, I. Holder, D. Kimberley, M. Bragg, S. McInally, A. Weiss and T. Williams, Conquering Complexity, London: TSO, 2005.

[6] International Organization for Standardization, *ISO/IEC/IEEE 42010:2011(E) Systems and software engineering - Architecture description,* IEEE, 2011.

[7] Object Management Groub, "What is SysML?," [Online]. Available: http://www.omgsysml.org/what-is-sysml.htm. [Accessed 7 10 2019].

[8] SysML Forum, "How should SysML be applied to an MBSE project? How is SysML commonly abused?," [Online]. Available: https://sysmlforum.com/sysml-faq/sysml-usage-modes.html. [Accessed 14 10 2019].

[9] M. Follmer, P. Hehenberger, S. Punz and K. Zeman, "Using SysML in the Product Developement Process of Mechanical Systems," in *International Design Conference - Design 2010*, Dubrovnik, 2010.

[10] A. Morkevicius , A. Aleksandraviciene, D. Mazeika, L. Bisikirskiene and Z. Strolia, "MBSE Grid: A Simplified SysML-Based Approach for Modeling Complex Systems," Nomagic, 2017.

[11] T. Costa, A. Sampaio and G. Alves, "On the Use of SysML in System of Systems Design," 2009.

[12] I. Ozkaya, "Representing Requirement Relationships," in *First International Workshop on Requirements Engineering Visualization*, 2006.

[13] G. Barbieri, K. Kernschmidt, C. Fantuzzi and B. Vogel-Heuser, "A SysML based design pattern for the high-level development," in *19th IFAC World Congress*, Cape Town, 2014.

[14] A. Salado and p. Wach, "Constructing True Model-Based Requirements in SysML," *Systems,* 2019.

[15] J. G. Lamm and T. Weilkiens, "Funktionale Architekturen in SysML," in *Gesellschaft für Systems Engineering*, München, 2010.

[16] K. E. T. Kernschmidt, Interdisciplinary structural modeling of mechatronic production systems using SysML4Mechatronics, München, 2019.

[17] J. Holt, S. A. Perry and M. Brownsword, Model-Based Requirements Engineering, London: Institution of Engineering and Technology, 2012.

[18] L. Lemazurier, V. Chapurlat and A. Grossetête, "An MBSE Approach to Pass From Requirements to Functional Architecture," in *IFAC (International Federation of Automatic Control)*, 2017.

[19] F. Mhenni, J.-Y. Choley, O. Penas, R. Plateaux and M. Hammadi, "A SysML-based methodology for mechatronic systems architectural design," *Advanced Engineering Informatics,* no. 28, pp. 218-231, 2014.

[20] S. Nejati, M. Sabetzadeh, C. Arora, L. C. Briand and F. Mandoux, "Automated Change Impact Analysis between SysML Models of Requirements and Design," 2016.

[21] R. Baduel, M. Chami, J.-M. Bruel and I. Ober, "SysML Models Verification and Validation in an Industrial Context: Challenges and Experimentation," in *European Conference on Modelling Foundations and Applications*, Toulouse, France, 2018.

[22] V. Saleshi, G. Florian and J. Taha, "IMPLEMENTATION OF SYSTEMS MODELING IN CONSIDERATION OF THE CONSENS APPROACH," in *INTERNATIONAL DESIGN CONFERENCE*, 2018.

[23] M. Nikolaidou, A. Tsadimas, N. Alexopoulou and D. Anagnostopoulos, "Employing Zachman Enterprise Architecture Framework to Systematically Perform Model-Based System Engineering Activities," in *Proceedings of the 42nd Hawaii International Conference on System Sciences*, 2009.